WORK BREAKDOWN STRUCTURE

&

TASKING DESCRIPTION

Table Of Contents

TENA PROJECT TASK STATEMENT

-

PROJECT DESCRIPTION:

There is an increasing need to prepare for a significantly expanded capability to transfer various types of information in near-real time between geographically separated test and training locations in order to test modern weapon systems or provide for realistic training Activities. The Test and Training ENabling Architecture (TENA) Project will define an architecture and facilitate a structure to enable the 21st Century capability for this information transfer within the T&E and Training communities.

TENA will identify and/or develop the standards, protocols and network attributes necessary to develop and demonstrate a concept that meets the T&E and Training needs within a system that can achieve the communications, connectivity and interoperability in the most economical and effective manner, and to enable increased use of modeling and simulation as valid T&E techniques within the Major Range and Test Facilities Base (MRTFB). TENA will entail concepts that foster an open-system architecture, use advanced digital electronics developments, and exploit distributed interactive simulation techniques and commercial-off-the-shelf technologies, where applicable.

The Navy is the CTEIP Resource Manager for this project, and will establish a joint project office for the management of the project Activities. This project office will be the principle point of contact for the T&E and Training communities for matters related to information transfer, networks, interconnectivity and interoperability.

The architecture defined will be domain-specific to promote application interoperability and asset sharing in both the T&E and Training domains within reasonable technical, cost and schedule constraints. The architecture should be consistent with the DoD High Level Architecture (HLA) currently under development by the Defense Modeling and Simulation Office (DMSO).

The TENA project is a two-year, process-based project. Therefore, an overall project process definition will be developed by the Project Management Office (PMO) as a project road map. The elements of this process definition are defined by the TENA core team. In addition, a process for execution of each task area will be developed by the Integrated Project Teams (IPTs) and followed during the life of the TENA project. The TENA project will use the Defense Information Systems Agency (DISA) Domain Engineering Process as the basic process for developing the architecture. This technique provides a three-step evolution for the architecture definition and will integrate well with the IPT developed processes.

The TENA Project has been divided into three top-level task areas; Project Management, Architecture Definition and Transition Planning. Six IPTs were created to support these top level task areas. The Army is assigned task leadership and IPT responsibility for the Applications concept and Concept Validation tasks. The Navy is responsible for the PMO, and is IPT lead for Architecture Definition and Requirements. The Air Force has the IPT responsibility for Transition Planning. Task statements and a Work Breakdown Structure (WBS) are supplied below for these task areas. Service team members will be expected to support tasks other than those specifically identified as the lead tasks.

Work Breakdown Structure Tasking Definition:

1. **PROJECT MANAGEMENT** -- The Test & Training ENabling Architecture (TENA) project management team lead by Mr. Ed Dunn - Naval Undersea Warfare Center (NUWC), is responsible for planning, controlling and executing the TENA project.

    1. **PROJECT PLANNING --** The TENA project management team is responsible for the planing and controlling of the project Activities as described in the TENA project work breakdown structure (WBS). The key Activities include planning

schedules, on-line communications management and requirements, architecture and transition planning. The TENA project office will use project management software to monitor and control the progress of the project.

1. **ON LINE SERVICES PLANNING** -- Since the TENA project is based in a virtual office environment with IPTs located on different regions of the United States, on-line services is of critical bearing for the success of TENA. The goal is to have readily access to databases, decision support systems, object repository, library services, presentations and briefs, administrative proceedings and project coordination.

    1. **ON-LINE SERVICES STUDY --** The TENA project management office will conduct a survey to determine the optimal solution that will meet the team on-line requirements.

    2. **ON-LINE SERVICES IMPLEMENTATION --** The TENA management office will implement the chosen on-line service solution and will train if necessary the rest of the team.

    3. **ON-LINE SERVICES MANAGEMENT --** It is the responsibility of the TENA project management team to maintain the on-line services.

2. **REQUIREMENTS PROCESS PLANNING** - Requirements development is the initial key area for the TENA project. The TENA must be based upon valid and derived service requirements as well as the latest technology developed in government and industry. The development of these requirements is critical to the success of the overall project. The TENA project has identified two categories of requirements. Category I represents requirements based on known networking needs which currently exist or are planned at the various T&E and Training facilities. Category II involves requirements based on the collection of applications concepts. All services will support the task of requirements collection which will be lead by Mr. Brad Ennen of Naval Air Warfare Center, China Lake, CA.

3. **ARCHITECTURE PROCESS PLANNING --** This task area consists of evolving the TENA architecture based on the TENA requirements and other associated DoD architecture Activities. The Navy has the IPT lead for architecture development and Mr. Ed Dunn of the Naval Undersea Warfare Center is assigned as the IPT leader. Deliverables from the architecture development task area are: architecture development process draft, demonstration and final TENA architecture reports.

4. **TRANSITION PROCESS PLANNING --** As required by the TENA sponsor the Central Test & Evaluation Investment Program (CTEIP), a transition plan will be developed to transfer the TENA concept to a Service or DoD Agency for implementation once demonstrated and

approved. The plan should define an appropriate transition point, estimated costs for sustaining efforts, documentation and approvals needed prior to the transition. The Transition activity focuses on packaging and documenting ideas in such a way that it can be used by follow-on programs.

2. **PROJECT CONTROL --** The TENA PM will exercise control over the project to ensure success. Particular attention should be given to milestones and schedules, budget, deviations from project baseline, goals, procurement needs or resources.

3. **PROJECT REPORTING --** As required by the CTEIP, the TENA project manager is in charge of the creation and update of the project reports. These include, the Project Management Plan, CTEIP Monthly Status Report, Financial and Progress Reports. The project manager will collect appropriate feedback from the deputies as needed in order to create these reports.

    1. **PROJECT MANAGEMENT PLAN** -- The TENA project manager, Mr. Ed Dunn, has the responsibility to write and to review the Project Management Plan. Standard CTEIP procedures will be followed.

    2. **MONTHLY REPORTS** -- The TENA project manager, Mr. Ed Dunn, has the responsibility to provide monthly CTEIP financial and progress reports. These reports will comprise of up-to-date information gathered from reports due from the IPT leaders. Standard CTEIP procedures will be followed.

4. **USER ADVOCACY AND PROPONENCY** -- A key to success of any new system or idea is the acceptance by the user who will ultimately determine its success or failure. In order to achieve user acceptance it is important to collect and answer user concerns and when possible incorporate user ideas. This activity will serve to first, educate the user about TENA and its potential benefits, and then collect and incorporate user feedback.

    This activity will serve to educate potential users as to what TENA is and the potential applications which can be realized by linking training and test facilities. The draft Applications document will be distributed and a representative briefing prepared for use in this activity. Trips and teleconferences will be required for this activity; however, these trips will be consolidated as much as possible with other TENA data collection activities.

    1. **TENA BRIEFINGS SCHEDULING -** There are a number of other DoD architectural-related Activities which are engaged in work which could impact TENA or which should be influenced by the TENA requirements. The TENA project office is responsible for coordinating the TENA team participation in those Activities to the extent necessary to insure the interests of the TENA community are represented. In addition, these

meetings and conferences are considered to represent cost-saving opportunities for interacting with the organizations and individuals which have inputs to the TENA effort. The TENA project office will publish an on-line calendar with future briefings schedules.

2. **TENA CONSORTIUM --** The TENA Consortium will comprise a group of T&E and Training representatives who will bring essential information regarding requirements, validation, implementation and transition of the TENA. This group could represent future TENA compliant users and as such, their involvement is highly beneficial for the success of the project..

5. **FINANCIAL MANAGEMENT** -- The TENA project office will be responsible for all financial management of the project to the highest level.

2. **ARCHITECTURE DEFINITION --** The TENA project will define a domain-specific architecture to promote application interoperability and asset sharing in both the T&E and Training domains within reasonable technical, cost and schedule constraints. A fundamental principle in formulating TENA should be that of adopting and reusing all appropriate DoD, Service and commercial standards, protocols and architecture guidelines. The architecture should be consistent with the DoD High Level Architecture (HLA) currently under development by the Defense Modeling and Simulation Office (DMSO). To this end, the TENA Project Office will closely monitor the progress and evolution of DMSO HLA efforts. In addition, the TENA architecture should be maximally compatible with the Service's major C4I initiatives to the extend practicable. For these reasons, it is of paramount importance that a requirements analysis is conducted so as to focus the effort toward those key interoperability opportunities with the greatest valued-added potential to meet validated requirements. The architecture should be driven by validated requirements, however because of the rapidity in the expansion of both technical capabilities for and applications of information interconnectivity, current requirements must be modified by prudent projections of future needs to ensure the architecture is not obsolete by the time it can be implemented, in the TENA project this is called Applications concepts.

   The TENA project will follow the Defense Information Systems Agency (DISA) Domain Engineering Approach to define the architecture. This is a three-step approach in which iterations are made in order to optimize the final product.

   1. **DEFINE TENA STEP I** -- Early in the program, the architecture team will collect data from the Requirements and Applications Concept teams in order to develop one or more notional or draft architectures. Although requirements collection and analysis will not be complete at this point, the production of draft architectures will test the development process, including human and software tool interfaces. The draft architectures should identify major issues to further focus subsequent efforts. The team must be careful to avoid placing too much effort on the content of these first architectures, accepting that they may change dramatically. The focus must remain on process and early identification of risk areas.

1. **IDENTIFY DOMAIN I** -- The goal of this process is to compile essential criteria that will be used to scope and select an initial target domain on which to focus initial analysis and design efforts. This activity identifies and describes the technical conditions used specifically to help focus initial analysis efforts in a domain with a very broad scope. The analysis efforts must be structured to focus initial activities on available information, while adhering to the initial conditions set for conducting the analysis. Before analyzing the domain, it is essential that analysts acquire a general understanding of the domain. To accomplish this, information about the domain is collected in the following subprocesses:

- Identify information sources,
- Gather initial domain knowledge,
- Describe the domain, and
- Verify domain description.

    1. **IDENTIFY INFORMATION SOURCES** -- The goal of this process is to identify reliable domain expertise and documentation resources that will be beneficial in formulating an understanding of the domain. Domain resources will be used in various process activities for: soliciting, collecting and extracting relevant domain information; developing and verifying domain models and other valuable sources of domain information.

Description:

To conduct an effective domain analysis, reliable sources of information relevant to the domain must be identified and consulted. Domain analysts identify sources of domain information and extract the domain data from these sources to formulation understanding of the domain, leading to the development of useful domain products. The domain analyst’s perception of system behavior in the domain is dependent upon the reliability of these information sources.

The information sources which contribute to the TENA program can be grouped into five areas: Visionary Sources (2.1.1.1.1), Networking Sources (2.1.1.1.2), Applications Sources (2.1.1.1.3), Business Process Sources (2.1.1.1.4), and Related Architecture Sources (2.1.1.1.5). Lower-level activities of "Identify Information Sources" may be assigned to be executed by different portions of the TENA team, however each will follow the overall process as described below. There are different types of information sources available to domain analysts including: domain experts, requirements specifications, system documentation, independent research, training materials and classes, and related seminars. Domain experts, requirement specifications and functional process improvement information models are probably the best sources of domain information. Domain experts may be individuals that define requirements for developers, users or maintainers of the domain's systems. Domain experts can also be used in the domain analysis process to validate the analyst's perception of the domain. The domain analyst compiles and maintains a contacts list that identifies potential domain experts and a list of the possible systems documentation and other information for consideration during the domain analysis process.

1. **IDENTIFY VISIONARY SOURCES** -- This survey addresses future programs and future training and test concepts which apply to interoperability. A number of meetings, teleconferences, and workshops will be attended and TENA-like interoperability ideas collected. These requirements will be predominately from organizations, and other high level panels such as the Defense Science Board.

2. **IDENTIFY NETWORKING SOURCES** -- This survey addresses networking sources across the T&E and Training community.

3. **IDENTIFY APPLICATION SOURCES** -- This process will identify potential applications concepts sources across the T&E and Training community. Application Concepts is fundamental to developing TENA. The process will consist of identifying and compiling a set of application concepts which represent beneficial value to the various communities to be served by TENA. There are three such communities whose interests must be considered and who can provide useful application concepts:

1. The Testing community

1. The Training Community

1. The RD&A [Spell out] Community

We also believe that a systematic approach to developing application concepts is appropriate. This will be conducted in parallel with the process of collecting ideas from the above mentioned communities. This approach will be to locate (or develop if necessary) a taxonomy of resources at the various facilities and then systematically consider the networking requirements and architectural implications associated with sharing each category of resource. Next, the application concepts compiled from canvassing the communities will be analyzed and the resulting implications folded into the systematic analysis of sharing resources. Finally, at the completion of this process, both economic and useability analysis will be performed, leading to a final document which highlights the requirements which are most likely to impact the architecture.

1. **IDENTIFY BUSINESS SOURCES** -- This process will identify potential business sources across the T&E and Training community.

2. **IDENTIFY RELATED ARCHITECTURE SOURCES -** **-** This process will identify potential architecture sources across the T&E and Training community.

3. **LIST OF INFORMATION SOURCES** -- As potential sources are identified, a list will be created for traceability and for analysis purposes to include visionary, networking, application, business, and related architecture sources.

1. **GATHER INITIAL DOMAIN KNOWLEDGE** -- The goal of this process is to begin the collection of domain information from the information sources identified in 2.1.1.1.6, this will be used to develop an initial understanding of the domain of interest and as input to prepare a description of the domain in an appropriate level of detail.

Description:

In this process, the domain analysts define an initial set of information requirements that will facilitate their high-level perception of the domain based on characteristics associated with systems in the domain. Solicited high-level information applicable to the domain may consist of the following:

- Business Models,
- Brief System Descriptions,
- System Capabilities,
- Available Domain Models and Domain-Specific Software Architecture (s) (DSSAs),
- Identifiable Common Systems Patterns,
- Operational Environment,
- External Interfaces,
- Systems with similar capabilities,

- Known Reuse Opportunities, and
- Available system documentation.

In general, information extracted from domain sources should concentrate on *what* systems in the domain do (i.e., system capabilities and functionality), not *how* system capabilities are implemented. This is particularly true for initial data gathering. However, as domain analysis efforts proceed to lower-level detail, implementation of systems capabilities may provide alternative solutions to solve a common problem in the domain and should be noted in the Domain Knowledge Base.

-

-

Activities:

The activities below will be conducted by the TENA team and will be recorded in the Domain Knowledge Catalog (2.1.1.3). These key activities can be conducted concurrently.

- Use existing knowledge from previous and on-going analysis efforts, whenever possible, from a Domain Knowledge Base. Extract qualified domain analysis products from a reuse repository.
- Use the system questionnaire contained in Appendix **** for recording information gathered from contacts and program management offices.
- Distribute system questionnaires or conduct interviews with domain experts (such as, Program Executive Office (PEO), Technical Integration Manager (TIM), Functional Activity Program Manager (FAPM), and software developers) for initial system data.
- Collect initial questionnaire and survey responses from domain experts and verify data. Additional documentation and domain experts may be uncovered during these activities. As required, the source list should be updated to reflect changes.
- Develop a Domain Knowledge Catalog in an indexed format showing all relevant domain information needed to conduct the domain analysis effort. This catalogue depicts the sources type and the types of the documents found during the fact finding process. This catalogue includes pointers to or addresses of domain documentation, questionnaires and interviews with Domain Experts, and potential reusable assets from a reuse repository.
- Attend system development reviews and coordinate with current projects' documentation distribution activities.

- Identify support documentation and other domain information to be used in further analysis activities.
- Gather relevant information from other possible sources such as the library, technical seminars/classes, and independent research.
- Collect the Business Process Model (IDEF0) .
- Identify and gather high-level system interface and architecture data.
- Acquire information needed for the domain analysis and do a preliminary or high-level first-cut analysis. Concentrate on *what* systems should do, not *how* they do it.
- Provide a Domain Engineering orientation for domain experts, and explain the purpose and goals for conducting domain engineering.
- Start a filing system to keep all information in a central location, the Domain Knowledge Base.
- Identify problems in the domain sources and initiate process improvement in the form of feedback to streamline the Identify Information Sources process.
- Use the feedback identified in Scope Domain (2.1.2) to incorporate any lessons learned, anomalies or discrepancies discovered during the scoping effort.
- Develop system criteria that will be used to partition and group systems in the domain, according to obvious commonalities, for the domain analysis effort.

1. **GATHER VISIONARY KNOWLEDGE --** Visionary information will be collected during this process and will be compiled in the Domain Knowledge Catalogue.

    1. **APPLICATIONS CONCEPT DOCUMENT --** This document will contain a collection of the Applications Concept defined on 2.1.1.1.1. These application concepts will consist of a number of scenarios each written to represent a different application concept. The scenario approach will consist of a time line describing the various activities including real-time aspects as well as the non-real time aspects of the exercise. A set of generic scenarios will then be generated with the intention of fulfilling the goal of broad applicability accross the various communities.

    2. **GATHER NETWORKING KNOWLEDGE --** Networking information will be collected during this process and will be compiled in the Domain Knowledge Catalogue.

    3. **GATHER APPLICATIONS KNOWLEDGE --** Applications information will be collected during this process and will be compiled in the Domain Knowledge Catalogue.

    4. **GATHER BUSINESS PROCESS KNOWLEDGE --** Business Process information will be collected during this procedure and will be compiled in the Domain Knowledge Catalogue.

    5. **GATHER RELATED ARCHITECTURE KNOWLEDGE --** Related architecture information will be collected during this process and will be compiled in the Domain Knowledge Catalogue.

1. **DOMAIN KNOWLEDGE CATALOGUE** -- This catalogue depicts the sources and the types of the documents found during the fact finding process. It is important to maintain the catalogue's currency to ensure the validity of the artifacts. In addition, analysts should store this catalogue in a knowledge base. This will be a gathering of all the domain information in order to formulate a general or high-level understanding of the context and behavior of the domain. Solicited high-level information applicable to the domain may consist of the following: business models, brief system descriptions, system capabilities, available domain models and DSSAs, identifiable common system patterns, operational environments, external interfaces, systems with similar capabilities, known reuse opportunities, and available system documentation.

2. **DESCRIBE THE DOMAIN** -- The goal of this process is to develop a general description of the domain. The domain description characterizes the domain in terms of its subdomains and established relationships to other domains. The domain definition provides a basis of understanding before proceeding with the analysis. In addition, the domain description will be used to predict and assess the level of effort that will be required to conduct the analysis.

Description:

The key of conducting domain analysis is to understand the context of the domain to be analyzed. The analyst must describe, at an acceptable level of detail, the domain in which analysis will be performed.

The domain analyst uses data collected from reliable information sources and archived in the Domain Knowledge Catalogue to compile a robust description of the domain. The domain may be described in terms of its systems, system capabilities, system interaction, subdomains and interaction between subdomains. In most cases, the domain description includes the context and scope (boundaries) of the domain and its interaction with external entities. For graphical illustration, the domain analyst develops a context diagram depicting this external interaction.

To describe the domain, the analyst must understand the high-level composition of the domain. To accomplish this, the analyst attempts to partition the domain's systems into groups based on obvious system commonalities, common functionality or common capabilities. Partitioned groups or "families" of systems that share a unique set of commonalities in the domain are referred to as subdomains. The domain may be composed of many interrelated subdomains.

The domain description, which characterizes the type of systems and associated subsystems in the domain based on common system attributes, also describes its related subdomains. Each subdomain is characterized by a unique set of related attributes.

Activities:

- All of the key activities below are conducted by domain analysts with results recorded in a knowledge base for future process efforts. These key activities can be conducted sequentially.

- Define the domain's context. To describe the domain:

  (1) initially scope the domain to determine its objects;

  (2) determine the responsibilities of the domain;

  (3) determine the services provided by the domain;

  (4) identify the dependencies and relationships the domain has with external domains; and

  (5) determine the high-level functions (e.g., supply, medical, personnel) of the domain and their interactions.

- Construct a context diagram illustrating the interfaces between the domain and the external domains. Use existing system context diagrams as a foundation for developing the domain context diagram. Show data and control flows in the context diagram.

- Use the information provided by the system questionnaire, survey responses and other possible sources to develop a general description.

- Use the Domain Knowledge Catalogue as an index for referencing key documentation, such as the SRS or FD, in constructing context diagram and extracting information for a domain description.
- Identify obvious boundaries and interfaces between different groups of systems and subsystems identified.
- Determine the general categories or types of systems based on obvious system similarities, and group systems accordingly.
- Determine the general relationships between the different types or families of systems and subsystems identified.
- Develop a general description for the initial set of systems and related subsystems under investigation and describe the system relationships.
- Continuously refine and update the description to reflect the scope of the domain.
- Using the Feedback from Scope Domain (2.1.2) incorporate any lessons learned, anomalies or discrepancies discovered during the scoping effort.

Mechanisms used to facilitate process activities include: Domain Analyst, a Traceability Tool and a Knowledge Base Tool.

1. **VERIFY DOMAIN DESCRIPTION:--** The goal of this process is to verify that the description output from the previous process matches the actual domain. domain experts independently verify correctness of the context diagram and the domain description.

Description:

The domain description elaborates the problem domain as understood by domain analysts. In order to proceed to the next activity, domain experts must verify the domain description including the context diagram which depicts interfaces to external domains. Experts apply their knowledge of the domain as well as knowledge embodied in relevant artifacts to formalize the domain description. These artifacts include the available business process model (IDEF0) and data model (IDEF1X) developed by the business process improvement program. Domain experts check the domain description for consistency with the business process and data models.

Activities:

All the activities below are recorded in a knowledge base for future process efforts. These activities can be initiated sequentially.

- Identify problems or issues with constructive recommendations to refine the domain description.
- Determine whether the domain definition delineates the domain with

sufficient detail and clarity such that the customer, domain experts, and domain analysts have a common understanding of its scope.

- Use the available business process model (IDEF0) to verify whether all the processes found at the highest level are clearly stated in the domain description.
- Use the business rules model (IDEF1X) to check whether external relationships are represented in the context diagram.
- Discuss and explain any problems to the domain analysts. Review recommendations with analysts in order to expedite the refinement activity.
- Obtain assistance from the domain analysts for clarification of the domain description, as required.
- Using the Feedback from the process, Scope Domain (2.1.2), incorporate any lessons learned, anomalies or discrepancies discovered during the scoping effort.
- Record the results of the domain description verification into the Knowledge Base Tool for the next iteration or future reference. Verification results include problems and recommendation for refining the description.

1. **SCOPE DOMAIN** -- The primary object of this process is to identify a (portion of a-) domain on which to focus subsequent domain analysis efforts. Constraining analysis efforts to manageable domains yields a complete and detailed domain model and domain-specific software architecture delivered in a timely manner. By scoping the domain, multiple or concurrent domain analysis efforts can be initiated in order to expedite the development of the final domain analysis and design products.

The process of scoping the domain consist of the following subprocesses:

- Establish Scoping Criteria
- Define Domain Boundaries
- Verify Domain Scope

Each subprocess has a unique set of associated activities to accomplish the prescribed subprocess objective.

1. **ESTABLISH SCOPING CRITERIA** -- The goal of this process is to compile essential criteria that will be used to scope and to select an initial target domain on which to focus initial engineering

efforts. This activity identifies and describes the technical conditions used specifically to help focus initial analysis efforts in a domain with a very broad scope.

<u>Description:</u>

The success of the domain analysis effort is based on the conditions established to conduct the analysis process. Technical conditions should be established to balance the domain's scope against available resources to perform the engineering.

Scoping criteria are success predictors for the domain analysis -- a way to screed out domains that have a low chance of being successfully analyzed. Prioritization and weight assignment using formal methods (e.g., Analytic Hierarchy Process - AHP) are important for formalizing the scoping criteria.

<u>Activities:</u>

All the key activities below are recorded in a knowledge base for future process efforts. These key activities can be initiated sequentially.

- If possible, derive scoping criteria from reasonably successful domain analysis and design efforts. Feedback from past efforts will provide valuable input to developing scoping criteria. The following criteria have been used on past domain analysis efforts and may be used as a basis for evaluating and scoping other domains:

    1. **Number of Existing Systems** - the number of existing systems partially determines the amount of knowledge accumulated about a domain. If a domain has less than three systems, the potential for commonalities across the domain diminishes.

    1. **Future Systems Development** - This factor identifies the number of projected new systems and redesigned systems that can benefit from a reuse analysis of the selected domain.

    1. **Vertical Reuse** - Potential for sharing assets of a given application in similar applications within the scope of the *same* domain.

    1. **Horizontal Reuse** - Potential for sharing assets with or across multiple domains.

    1. **Reuse Expertise** - The amount of reuse and domain knowledge employed during the development of a system greatly affects the quality of extracted information and reusable assets. Development team experience acquired during development will aid in the communication of domain concepts, model validation, and general interaction with domain analysts and domain designers.

    1. **Stability** - A domain is stable if it experiences little change (e.g., additions, extensions, modifications, and deletions of

requirements) during the time period of concern.

1. **Participation** - Availability of systems and operational personnel for phone and personal interviews, and for model verification and DSSA validation. Willing participation and domain expertise is integral to conducting domain analysis and design efforts through data collection, model verification, and DSSA validation.

1. **Available Information** - In domain analysis, it is essential that quality domain information be available for data collection. The information will be used to gather and analyze system information. This domain information includes: business process and rule models, requirements, design, code, test, user documentation and working papers.

- Tailor the aforementioned scoping criteria to conform with the Verified Domain Description (2.1.1.5) identified in the process. For instance, the number of existing systems criterion would not apply to an unprecedented domain (new domain).
- Describe each scoping criterion in sufficient detail and ensure these criteria correspond with the domain description.
- Determine weights to attach to each criterion. Relative weights are assigned to criteria to emphasize their relative importance.
- Compile rationale supporting weight assignment to each criterion.
- Using the Feedback identified in the process, Analyze Domain (2.1.3), incorporate any lessons learned, anomalies or discrepancies discovered during the development of the domain models.

1. **DEFINE DOMAIN BOUNDARIES --** The goal of this process is to assess, select and describe the boundaries of the domain upon which to focus initial domain efforts. The domain boundaries will provide a starting point from which domain analysis efforts will expand. With a domain that has an extremely broad scope this process is essential to help focus the initial analysis effort. This may be considered an optional activity, if the domain is already well defined.

Description:

The success of the domain analysis depends on many factors. One of these determinants is *where to begin the analysis.* "Divide and conquer" is the underlying approach to conduct an effective domain analysis where the domain is very broad and there are limited resources. In these cases, limited resources dictate that analysis efforts concentrate on the initial area of interest within the domain which is expected to provide the richest area of commonality and differences. The results of the initial domain analysis efforts then provide a foundation from which the effort will continue to grow and embrace the remaining areas of the domain.

To focus initial domain analysis efforts, an initial area of interest must be determined. To accomplish this, selection criteria must be applied to each subdomain to compute a score that indicates a good starting point. Domain analysis efforts must be applied carefully for analysis, based on the maturity and stability of the organizations and activities within each subdomain, and on the planned emphasis the subdomain is to receive.

Activities:

All of the key activities below are conducted by domain analysts and results are recorded in a knowledge base for future process efforts. These key activities can be initiated sequentially.

- Build a Domain Boundary Matrix by applying the domain selection criteria to each family of systems.
- For each family of systems, determine a rating for each criterion established in the previous process (2.1.2.1).
- Compute the overall score for each family of systems by adding together the rates of all the criteria.
- Select the subdomains or family of systems that have the highest score for the initial domain analysis efforts. Prioritize subsequent domain analysis efforts based on scores.
- Provide rationale for assigning domain analysis priorities based on rates recorded in the Domain Boundary Matrix.
- Using the Feedback identified in the process, Analyze Domain (2.1.3), incorporate any lessons learned, anomalies or discrepancies discovered during the development of the domain models.

1. **VERIFY DOMAIN SCOPE** -- The goal of this process is to ensure the correctness and completeness of the subdomain identified in Identify Domain (2.1.1). This process is performed by domain experts to check whether appropriate weights were applied to each criterion and that the decision made to focus domain analysis efforts was objective, not subjective.

Description:

The success of domain scoping depends on its correctness and completeness. It is vital that domain boundaries be clearly defined to expedite the domain analysis effort and to maximize the identification of commonalities and differences. To achieve this objective, domain experts play an important role in providing an independent technical examination and verification of the bounds of the domain established in the Define Domain Boundaries process (2.1.2.2).

Domain experts check for consistency in the computation of weight values and that these values are properly applied to each criterion. If needed, domain experts should obtain assistance from domain analysts in clarifying any problems or issues arising during the verification process. These problems should be recorded as part of the Domain Boundary Refinements with appropriate recommendations and solutions. To accelerate these corrections, domain experts should review proposed changes with domain analysts by explaining the purpose of the refinements and justifying the recommendations.

In conjunction with this step, Management Direction concentrates on the feasibility of the domain analysis effort by assessing the depth and width of the domain. Once domain boundaries are accurately defined, elaborated, satisfy management requirements and pass domain experts' testing, the domain scope is verified and approved. Using the Verified Domain Scope Report (2.1.2.3.1), domain analysts can initiate the next phase, Analyze Domain (2.1.3).

Activities:

The activities described below are conducted primarily by domain experts, with the assistance of domain analysts, where required. Results are recorded in a knowledge base for future process efforts. These activities can be initiated sequentially:

- Using the Domain Boundary Matrix, verify that all the weights are properly and consistently applied to each criterion.
- Check for subjective decisions made in assigning values. Get assistance from domain analysts for clarifications or reasons for the derivation of selection criteria weight values.
- Identify problems, issues, and recommendations for the matrix.
- Discuss and explain problems with domain analysts. Review recommendations with analysts in order to speed the refinement activity.
- Suggest revisions to the domain description for conformance with the defined domain boundaries
- Obtain assistance from domain analysts in ensuring that the Verified Domain Scope meets management requirements established under Management Direction.

- Domain analysts use the Feedback identified in the process, Analyze Domain (2.1.3), to incorporate any lessons learned, anomalies or discrepancies discovered during the development of the domain models.

    1. **DOMAIN SCOPE REPORT --** This report will contain information compiled during the Scope Domain Process (2.1.2).

1. **ANALYZE DOMAIN** -- The objective of domain analysis is to identify, derive, organize, abstract, and represent the body of knowledge of a particular domain. This body of knowledge is represented by a domain model. A domain model is similar to the system model derived in object-oriented requirements analysis; however, the domain model includes reuse guidance in the form of adaptation requirements, discarded alternatives, rationale, lessons learned, and identification of systems supplying or consuming the domain products. This information is key to constructing requirements for future systems within the domain. The domain model provides input to both the domain designer and the application software analyst.

The process of analyzing a domain consists of the following subprocesses:

- Organize & Synthesize Problem Space Information,
- Identify Commonalities,
- Determine Common Objects Adaptation Requirements, and
- Verify Domain Model.

These subprocess can be performed in sequence, although experience has shown that iterations of the process may produce greater detail and understanding, thereby increasing reuse potential. Information gathering tends to extend almost throughout, as some documents may reference other publications necessary for, or helpful to, the analysis effort. The identification of adaptation requirements often takes place each time a commonality is discovered.

Each subprocess has a unique set of associated activities to accomplish its prescribed objective. Subprocess activities provide guidance and a framework for conducting the process. Innovative methods are also encouraged to supplement existing activities.

As illustrated in Analyze Domain (2.1.3), each subprocess has: an assigned set of inputs which are transformed into associated outputs; controls that govern the subprocess activities; and mechanisms that indicate the means by which the subprocess activities are performed.

The following sections describe the subprocesses necessary to properly analyze a chosen domain. Each subprocess is described in terms of its basic objective, description, associated activities, controls, mechanisms, inputs, outputs, process improvement efforts, lessons learned (if any), and examples (if any).

1. **ORGANIZE & SYNTHESIZE PROBLEM SPACE INFORMATION** -- The goal of this process is to merge and categorize existing systems analysis information into a Problem Space Catalogue. This catalogue provides an index of all relevant software engineering information (e.g., Concept document, requirements document (SRS or FD), interface requirements specification (IRS), business models, existing source code, technical literature). The problem space catalogue is used by other process activities to easily locate domain information. Specific systems analysis information solicited from current development efforts include: Object-Oriented (OO) Diagrams, Data Flow Diagrams (DFDs), Entity-Relationship Diagrams (ERDs), Entity-Relationship-Attribute Diagrams (ERADs), Global Data Model/Logical Data Model (GDM/LDM), Business Process Models (IDEF0), Data Models (IDEF1X), State Transition Diagrams (STDs), Time Dependency, And Other Pertinent Analysis Diagrams.

Description:

To formulate an accurate perception of the domain, it is essential that reliable, detailed domain information be identified, gathered and evaluated for future use. Valuable domain information collected by analysts will be used during ensuing activities to help determine domain commonalities and differences.

The primary source of domain information is the domain expert. Domain expert involvement is critical for the identification of the fundamental reusable software assets which are needed for future applications/development within the domain. Individuals experienced in defining requirements or developing systems in the domain are an initial source of information for identifying underlying system requirements, capabilities and functionality. Ideally, domain experts who are focusing on the front-end of the software life-cycle are indispensable in soliciting analysis information and verifying the domain model.

Documentation plays a key role in system analysis; unfortunately, it is often out-of-date and/or incomplete. System designs, user manuals, other documentation and source code for recent system development in the domain are also targets for detailed information gathering. Specific systems analysis information solicited from current development efforts include:

- Diagrams: Object-Oriented (OO) Diagrams, Data Flow Diagrams (DFDs), Entity-Relationship Diagrams (ERDs), Entity-Relationship-Attribute

Diagrams (ERADs), Global Data Model/Logical Data Model (GDM/LDM), Business Process Models (IDEF0), Data Models (IDEF1X), State Transition Diagrams (STDs), Time Dependency, and other pertinent analysis diagrams.

- Documentation: SRS or FD, concept document, user's manual, and other important analysis documentation

If source code is available from legacy systems, a reverse engineering technique is employed to analyze the program. Existing source code is reverse-engineered to uncover system entities and structures so that analysts may acquire a better understanding of system capabilities and behavior. The procedures produce representations of the program in graphical form, such as dependency diagrams, structure charts and architectural diagrams. The aim of reverse engineering is to recover the design of existing software systems, then abstract recovered design information and express the design in a form appropriate for display and manipulation by automated CASE tools.

Once collected, domain information is coarsely evaluated by the analyst, based on the quality of knowledge provided and its relevance to the current analysis focus. Immediate analysis requirements determine which domain information will provide the most benefit to current analysis efforts. However, all information should be catalogued and stored (for media or soft-copies) in the Knowledge Base for future reference and analysis.

<u>Activities:</u>

All of the key activities below are conducted by domain analysts. Domain experts are interviewed and the interview results are recorded in a knowledge base for future process efforts. These key activities can be initiated sequentially.

- Interview personnel responsible for determining system requirements for the family of systems that fall within the selected domain. Concentrate the interview on extracting or identifying any underlying or common set of requirements that are shared by all systems that fall within the domain scope. In addition, ask questions relating to system differences within the selected domain.

- Using the Domain Knowledge Catalogue developed in the process, Gather Initial Domain Knowledge (2.1.1.2), as a framework, expand the list with more detailed analysis information focusing on system documentation (e.g., SRS, FD, software concept document, user's manual), system source code, system design decisions, and other system information. Collect (from domain experts) and organize these documents to get a better understanding of the system(s).

- Extract a complete set of graphical system representations from the collected documentation for each system within the domain. Graphical system representations are formal models: OO diagrams, DFDs, STDs, ERDs, ERADs, GDM/LDMs, structure charts, etc.

- Reverse-engineer existing source code to uncover system entities and structures so that analysts may acquire a better understanding of system capabilities and behavior. This is accomplished by producing representations of the program in graphical form, such as dependency diagrams, structure charts or architectural diagrams.

- Develop a Domain Analysis Catalogue that indexes all compiled, organized, and synthesized domain information, including the reverse-engineered products.

- Using the Feedback identified previous processes incorporate any lessons learned, anomalies or discrepancies discovered during the development of the complete DSSAs.

- Identify problems encountered during the development of the Domain Analysis Catalogue which might result in modification of the domain scoping.

1. **ORG. & SYN. NETWORKING REQUIREMENTS --** Follow the procedures described in 2.1.3.1 concentrating in networking requirements.

2. **ORG. & SYN. APPLICATIONS REQUIREMENTS --** Follow the procedures described in 2.1.3.1 concentrating in applications requirements.

3. **ORG. & SYN. BUSINESS PROCESS REQUIREMENTS --** Follow the procedures described in 2.1.3.1 concentrating in business process requirements.

    1. **REQUIREMENTS DOCUMENT** -- This document will contain a collection of the findings during the Requirements Process, it will feed the Requirements Traceability Matrix.

4. **DETERMINE REQUIREMENTS TRACEABILITY --**

    1. **TRACEABILITY MATRIX** -- An important concept for TENA will be the ability to trace requirements to programs to service unique test and training needs. This will be compiled in a traceability matrix. The requirements traceability matrix is required to support the requirements validation process. The matrix will indicate the type

of requirement, time frame, mission need statement (MNS) or operational requirement document (ORD) if applicable, service, command and individual program point of contact, telephone number and remarks.

5. **DETERMINE REQUIREMENTS BENEFITS --** The list of potential requirements and applications concepts will initially be gathered without particular regard to any consideration of the benefits associated with the concept. After compiling and distilling a representative list, a benefit analysis will be performed. In order to minimize the effort associated with such an activity, the analysis will concentrate on the relative benefit associated with each requirement. The idea is not to cost justify any particular concept but to simply rank them so that the relative importance can be conveyed to the architecture team and major drivers can be identified. Factors to be considered include but are not limited to: cost, benefit, feasibility, usability, and reusability.

    1. **RELATIVE BENEFITS STUDY DOCUMENT** -- This document will contain a description of the requirements and the weight factors used to determine the requirement relative benefit.

6. **VERIFY REQUIREMENTS --**

    1. **DOMAIN REQUIREMENTS STEP 1 DOCUMENT** -- The resulting data sets will provide a distillation of the requirements along with traceability in support of the TENA requirements task as well as a series of joint interoperability ideas from applications concepts task. After these requirements are consolidated, they will be placed into a database for ease of access and review. An important concept for TENA will be the ability to trace requirements to programs to service unique test and training needs.

1. **IDENTIFY COMMONALTIES** -- This effort determines the amount of reuse potential to be realized from the domain engineering effort. Commonalities can be in the form of objects, classes, functions, processes, entities, relationships, data elements, and so forth. The purpose is not necessary to find exact matches but similarities. Commonality can be identified in a number of ways, such as:

- Identification by Domain Experts - Input is necessary from individuals such as system users, functional proponents, commanders, and other personnel who work with and understand the problem domain in question. Their knowledge of repeated processes and data manipulation algorithms is extremely valuable. Different types and levels (i.e., skill area, rank) of domain experts provide multiple perspectives of commonality.

- Examination of Analysis Documentation - All analysis documentation, such as object diagrams (interaction, inheritance, aggregation, etc.), data flow diagrams, state transition diagrams, and business process and data models, are potential sources for the modeling of the requirements of the domain. As discussed in [CAMP90], software components can exist at different levels of abstraction. Similarities can be found at the smallest component level, at a CSC level, or at an architectural level. Often, the same information is represented with different formats, terminology, and in different groupings.

- Examination of Design Documentation - Design documentation portraying previous system solutions can provide commonality information through the study of the use of generics, templates, and other reuse mechanisms. Often, a system's analysis products are in a functional decomposition format. If the corresponding design employed OOD methods, identification of problem space objects in the design can also be useful to the domain analysis process. This constitutes a reverse engineering of the design. The analyst must, however, be alert to avoid introducing solution space information into the problem space and remain focused on only the problem space objects found in the design.

As in object-oriented analysis, domain commonalities are generalized into classes that represent abstractions of their respective set of objects. This process is not strictly top-down (i.e., without regard for existing assets) in nature, nor is it purely bottom-up (i.e., basing analysis only on existing assets). The goal is to satisfy future requirements with as many existing assets as practicable.

The commonality identification progression does not just proceed from a high level of abstraction to the lowest. Instead, there is an undulation as new information discovered in the lower levels of detail affects parent classes/objects. There is an iterative shaping of classes as new similarities are discovered.

The process of identifying commonalities consists of the following subprocesses:

- Identify Objects & their Relationships,

- Determine Behaviors,

- Identify Constraints,

- Develop Domain Common Objects Model, and

- Verify Domain Common Objects Model.

These subprocesses can be performed in sequence, although experience has shown that iterations may produce greater detail and understanding, thereby increasing reuse potential. Information gathering tends to extend almost throughout, as some documents may reference other publications necessary for, or helpful to the analysis effort. The identification of adaptation requirements often takes place each time a commonality is discovered.

Each subprocess has a unique set of associated activities to accomplish the prescribed subprocess objective. Subprocess activities provide ample guidance and a framework for conducting the process. Fresh, innovative methods are also encouraged to supplement existing activities.

In Identify Commonalities (2.1.3.2), each subprocess has: an assigned set of inputs which are transformed into associated outputs; controls that govern the subprocess activities; and mechanisms that indicate the means by which the subprocess activities are performed. The following sections describe the subprocesses necessary to identify the domain for analysis. Each lowest level subprocess is described in terms of its basic objective, description, and associated activities.

1. IDENTIFY OBJECTS & THEIR BEHAVIORS(RELATIONSHIPS) --

The primary focus of object identification is to capture common objects, associated characteristics in the form of attributes and services, and relationships consisting of object structures and their connections. Object-oriented modeling techniques are recommended as the foundation of the domain model because it helps to [COAD91e: 3]:

- **Tackle more challenging problem domains.** OOA brings extra emphasis to the understanding of problem domains.

- **Improve analyst and problem domain expert interaction.** OOA organizes analysis and specification using the methods of organization which pervade people's thinking.

- **Increase the internal consistency of analysis results**. OOA reduces the bandwidth between different analysis activities by treating Attributes and Services as an intrinsic whole.

The object-oriented world is more disciplined than that of conventional structured techniques. It leads to a world of reusable classes, where much of the software construction process will be the assembly of existing well-proven classes [MART92: 31]. There are various object-oriented paradigms that can be used to represent the object-relationship model. This document will not endorse a particular object-oriented technique but it will provide the reader with essential and general object-oriented constructs to develop the object-relationship model.

The process of identifying commonalities consists of the following subprocesses:

- Determine Composition,
- Identify Structure, and
- Determine Connection.

These subprocesses can be performed in sequence. Experience has shown, however, that iterations may produce greater detail and understanding, thereby increasing reuse potential. Information gathering tends to extend almost throughout, as some documents may reference other publications necessary for, or helpful to the analysis effort. The identification of adaptation requirements often takes place each time a commonality is discovered.

1. **DETERMINE BEHAVIORS --** The goal of this process is to identify object behaviors by understanding and determining the way each object interacts, functions, responds, or performs. Each object is considered in isolation from others. The behavioral model yields three basic components[EMBL92: 9-10]:

1. The states that each object exhibits throughout its existence;

1. The conditions that cause an object to make the transition from one state to another; and

1. The actions performed by, or on, an object in various states and transitions.

This process is optional if the domain does not exhibit any real-time characteristics.

<u>Description:</u>

Yourdon characterizes a behavioral model as "the required behavior of the insides of the system necessary to interact successfully with the environment" [YOUR89: 326]. This characterization applies to structured analysis, but the same concept can be employed in the object-oriented analysis paradigm.

Behavior is how an object acts and reacts, in terms of its state changes and message passing. In other words, "The behavior of an object is completely defined by its actions." [BOOC91: 80] So, a behavioral model for an object is similar to a job description for an object. Both describe what an object must do within a system. [EMBL92: 60] A behavioral model encompasses the object states, triggers and transitions, and actions.

An object *state* represents an object's status, phase, situation, or activity. While identification of object states is an essential part of behavior modeling, it is of little value without examining how one state relates to one another in the dynamic context of system operation. The process of changing the state of an object is called a *transition*. The events and conditions that activate state transitions are called *triggers*. In addition to states and transitions among states, a behavioral model exhibits the performance of an object by modeling its actions. An *action* may cause events, create or destroy objects and relationships, observe objects and relationships, and send or receive messages. [EMBL92: 60-62]

For more explicit information regarding behavioral modeling, readers can refer to OOA literature and books, such as: *Object-Oriented Systems Analysis-A Model Driven Approach*, by Embley et al.; *Object-Oriented Design with Applications*, by Booch; *Object-Oriented Modeling and Design*, by Rumbaugh et al; and other OOA publications cited in Appendix C.

Activities:

All of the key activities below are conducted by domain analysts using an object modeling tool and a traceability tool. Results are recorded in a knowledge base for future efforts. These key activities can be initiated sequentially.

- Use the object relationship model as the foundation to determine the object events in terms of states, triggers and transitions, and actions.
- Identify object states by looking at how an object behaves in a domain. A state represents an object's status, phase, situation, or activity. [EMBL92: 60]
- Identify triggers and transitions by determining the expected events and system conditions that activate a change of state for an object.
- Identify the actions that an object performs. An action may cause events, create or destroy objects and relationships, observe objects and relationships, and send or receive messages. [EMBL92: 62]
- Record results of object behaviors identification in the Class/Object Requirement Specification by entering information into the behavioral fields. NOTE: Composite Event Names represent states of an object at the compound level, whereas Primitive Event Names denote states of an object at the atomic level. For example, if the object is a Purchase_Order and the possible states or events for this object are Cut, Filled, Shipped, and Closed with Shipped being the Composite Event, then a Primitive Event for Shipped might be Logged and Send.

1. **IDENTIFY CONSTRAINTS** -- This process will determine the required qualities and limitations of an object. Each element in the domain model may be annotated with additional constraints, representing conditions that must be met at particular points in time, or

over specific periods of object activity. Constraints are used to denote the limitations of the domain on its objects and their behavior.

Object constraints denote the limitations of the domain on its objects and their behavior. Constraints serve to rule out anomalous states [LOND86]. They represent a general capability to declare conditions that must be true for a particular element or set of related elements. There are a number of possible forms constraints may take, including:

- Existence of attribute values or relations,
- Cardinalities of attributes or relations,
- Constraints on allowable values of an attribute,
- Conditions relating multiple attribute values that constrain the values that these attributes may hold,
- Preconditions to an operation,
- Postconditions to successful completion of an operation, and
- Temporal relationships of objects and operations. [GILR89: 63]

Activities:

All of the key activities described below are conducted by domain analysts using an object modeling tool and a traceability tool. Results are recorded in a knowledge base for future efforts. These key activities can be initiated sequentially.

- For persistent objects, identify the duration or time span of the attribute values or relations.
- Determine cardinality constraints of attributes or relations by specifying the upper and lower bounds of the cardinality.
- Define the allowable values of an attribute.
- Identify conditions that relate multiple attribute values which constrain the values these attributes may hold.
- Determine the preconditions and postconditions to an operation.
- Capture the temporal relationships of objects and operations.

1. **DEVELOP DOMAIN COMMON OBJECTS MODEL -** - The goal of this process is to consolidate and finalize the domain common objects model. The integrated domain common objects includes all the information identified and

recorded in the aforementioned process within the Identify Commonalities (2.1.3.2) process.

In addition, rationale and tradeoffs, classification terms, and any pertinent object characteristics (e.g., concurrency, external interfaces) are defined and documented to complete the domain common objects model diagrams and specifications.

Description:

The Domain Common Objects Model is not complete until rationale and tradeoffs for each class and object are documented, common objects classification terms are defined, and class and object characteristics are fully specified. This process brings all the information assimilated and generated during the identification of objects and relationships, determination of object behaviors, and identification of object constraints together to form the final basis of the common objects model.

The Object-Relationship Model, Object-Behavioral Model, and Object Constraints are consolidated to produce the formal common objects model. During this process, analysts finalize the common objects model diagrams by ensuring that all classes and objects are characterized with attributes and operations, relationships are established with appropriate structures and connections, and the required object behaviors and constraints are portrayed and documented. In addition, analysts complement the Class/Object Model Specification by capturing rationale and trade-offs for each class and object to justify the class and object existence. Analysts also develop the initial classification terms that will be used to qualify and retrieve reusable software assets.

A faceted classification scheme is employed to represent the classification terms. A faceted classification scheme identifies a set of facets, or categories, applicable to assets within a reuse library. These facets characterize the asset by identifying distinguishing attributes (in the form of facet terms) that can be used to classify, search, understand, evaluate, and select reusable software assets. An asset classification is determined by the aggregate, or synthesis, of its attributes for all facets.

Activities:

All of the key Activities below are conducted by domain analysts using an object modeling tool and a traceability tool. Results are recorded in a knowledge base for future process efforts. These key Activities can be initiated sequentially.

- Integrate the Object-Relationship Model, Object-Behavioral Model, and Object Constraints diagrams by ensuring that all classes and objects are characterized with attributes and operations, relationships are established with appropriate structures and connections, and required object behaviors and constraints are portrayed and documented.

- Construct classification terms using a faceted classification scheme.

1. Compile a set of terms that are unique to the domain (i.e., system acronyms, synonyms, etc.).

1. Class structure should be used as the initial basis for the domain categorization. Terms used to identify class and class characteristics should provide substantial input to deriving a domain classification scheme.

1. Derive a preliminary classification scheme based on component types and object specifications.

1. Create the domain classification scheme from available domain terms. Provide a description for each domain-related term. Assess term descriptions to eliminate redundancy and ambiguities. Use standard terms.

The following Activities represent additions or changes made to the Class/Object Requirement Specification.

- Identify rationale and trade-offs for each class and object. Include this and a full description of each class and object in the CORS. Record the results of the rationale and tradeoffs (e.g., issues raised, alternatives considered and reasons for choosing one alternative over others) in a Knowledge Capture Tool.

- Define required concurrency aspects for each class and object and elaborate in the CORS. [Optional]

- Specify external interfaces with associated external class or object names.

- Develop Logic/Algorithms for every class and object service. As a framework, use Yourdon's *Structured English* technique to describe the class and object services' logic/algorithms. NOTE: According to Yourdon, *Structured English* is a subset of the full English language with some major restrictions on the kind of sentences that can be used and the manner in which sentences can be put together. It is also known as PDL (for Program Design Language) and PSL (for Program Statement Language or Problem Specification Language). [YOUR89: 206-207].

The Activities below apply to the incorporation of refinements identified in the verification process. These Activities can be initiated sequentially.

- Review recommended changes with domain experts to expedite the refinement process.

- Use the Domain Common Objects Model Refinements output from the Verify Domain Common Objects Model process (2.1.3.2.5), to incorporate errors and discrepancies identified in the verification process.

1. **VERIFY DOMAIN COMMON OBJECTS MODEL** -- . This activity ensures correctness and completeness. Domain experts independently verify the common objects diagrams, the class/object requirement specification, and the classification terms.

Description:

The domain model of common objects represents the common aspect of the domain model. To proceed to the next process, which is to identify differences in the domain requirements, domain experts must independently verify the domain model of common objects developed by domain analysts. The domain model of common objects includes common objects illustrations, specifications, rationale and trade-offs, and common objects classification terms.

It is vital that domain experts have systems analysis experience, specifically in the object-oriented analysis paradigm. With this background, experts can comprehend the object-oriented notation used by domain analysts in developing the domain common objects model.

There are two levels of verification. For the first level, experts focus on whether the domain model of common objects conforms with the business process model (IDEF0) and data model (IDEF1X). This ensures that entities, entity relationships, and Activities are captured in the domain model of common objects. In addition, traceability is established from the domain model of common objects to the business process and data models. For the second level, experts check for complete representation and description of objects, object relations, object behaviors, and object constraints. Specifically, experts verify the encapsulation of the common objects by looking at common attributes, services, and relations.

Activities:

All of the key Activities described below are conducted by domain analysts and the domain experts. Results are recorded in a Knowledge Base Tool for future efforts. These key Activities can be initiated sequentially.

- Determine if all the entities and their relationships depicted in the IDEF1X diagrams are represented in the domain model of common objects. Ensure that appropriate traceability is established from each class and object to the entities defined in the data model.

- Verify that all the Activities illustrated in the IDEF0 diagrams are captured in the model. Ensure that appropriate traceability is established from each class and object's services to the Activities defined in the business model.

- Check for complete representation of objects, object relations, object behaviors, and object constraints.

- Check the encapsulation of the common objects by looking at common attributes, services, and relations.
- Check to ensure all the fields of the Class/Object Requirement Specification are entered and that the fields conform with the common objects diagrams.
- Identify problems or issues with constructive recommendations to refine the domain common objects model.
- Discuss and explain any problems with domain analysts. Review suggestions to correct these problems with analysts in order to expedite the refinement activity.
- Use the Feedback identified in the process, *Develop DSSA Guidelines* (Node A4.5 in Figure 4-19), to incorporate any lessons learned, anomalies or discrepancies discovered during the development of the complete DSSAs

1. **DETERMINE COMMON OBJECTS ADAPTATION REQS** -- The goal of this process is to identify the differences among domain common objects, i.e., Adaptation analysis. Adaptation analysis is critical in deriving a TENA and component library that can adapt to future system needs. This analysis of required adaptation may be based upon mission, threat, domain, or system planning information [GILR89]. Adapting requirements is accomplished either by annotating the class/object requirement specification, or by defining the overall domain adaptation requirements and allocating them to specific common objects. These factors can be identified concurrently with commonality identification.

<u>Description:</u>

Adaptation requirements identify the different applications of the common capabilities. Just as commonality identification is necessary to establish a grounds for reuse, adaptation identification is necessary to tailor the information to particular needs.

Adapting requirements is accomplished either by annotating the Class/Object Requirement Specification, or by defining the overall domain adaptation requirements and allocating them to specific common objects. These factors can be identified concurrently with commonality identification.

The techniques to represent adaptation can take many forms. The generalization-specialization feature of OOA provides for subclasses where different configurations can be represented. The CORS have adaptation sections in the class/object, attribute, and operation sections. State transition and/or structured logic representations can also identify tailorable needs by modified notation.

Often, another perspective on commonality may be realized in this step, causing a refinement to a class content or hierarchy.

In some cases, certain capabilities may be mutually exclusive and cannot exist together in the same system. This cardinality can be represented by an instance connection that denotes a zero-to-zero relationship between objects/classes.

Activities:

All of the key Activities described below are conducted by domain analysts using an object modeling tool and a traceability tool. Results are recorded in a knowledge base tool for future efforts. These key Activities can be initiated sequentially.

- To determine appropriate adaptation requirements, use the following factors as a framework:

  1. Flexibility in operation,
  2. Mission adaptation (needs, threats, etc.),
  3. Environment/site adaptation,
  4. Platform adaptation,
  5. User adaptation, and
  6. New technology adaptation.

- Observe trends from an analysis of changes or evolutions of previous systems.
- Update the domain common objects diagrams by adding subclasses and their characteristics (attributes and services) to existing class constructs. Use a generalization-specialization structure to represent the new subclasses.
- Identify any adapted object connection, behavior and constraints.
- Describe adaptation capabilities for each class and object, attribute, and operation.
- Capture classification terms for the adapted model and merge with the common classification terms.
- Identify rationale and tradeoffs for each object adaptation. Use a Knowledge Capture tool to record this data. Rationale and tradeoffs describe issues raised during development, alternatives that were considered, and reasons for choosing one alternative over others.
- Consolidate the Verified Model of Common Objects and their documented

adaptations to form the complete domain model.

- Develop guidelines for the domain model describing how to use the model in a new system development. The guidelines should include the following:

    1. Notation (e.g., Coad/Yourdon, Rumbaugh, Colbert) or nomenclature used to develop object-oriented diagrams;

    1. Format of diagrams and specifications (i.e., automated tools);

    1. Definition and scope of the domain;

    1. Documentation references; and

    1. Potential usage of domain model.

1. **VERIFY DOMAIN MODEL** -- The goal of this process is for domain experts to independently verify the domain model (i.e., Object-oriented diagrams, specifications, classification, guidelines, and rationale and trade-offs). Prior to this process, both commonalities and differences of projected future system instances are well understood for the family of systems in the application domain. Verification of the domain model can be achieved in the following ways: inspection, prototyping, and simulation.

<u>Description:</u>

The domain model must be verified to establish a reliable baseline**.** The business process and data models, if available, should be used as the "doctrine" against an operational scenario. Verification can be accomplished in the following ways:

- **Inspection** - A team of domain experts examines the model to verify completeness and correctness. Preferably, some of the domain experts have not taken part in the information gathering, thereby providing an independent review. Operational scenarios can be used at this step to assure the correctness of the model.

- **Prototyping** - A small set of capabilities can be implemented using a prototyping tool/language. The prototype would then be executed.

- **Simulation** - A simulation model can be constructed to refine the object model and to explore adaptation mechanisms. Simulation provides an excellent means for analyzing the effects of future system environment changes (i.e., mission, threat, theater, etc.) and for performing "what-if" analyses. [GILR89: 69]

Activities:

All of the key Activities described below are conducted by domain analysts and domain experts. Results are recorded in a Knowledge Base Tool for future efforts. These key Activities can be initiated sequentially.

- Use the business process (IDEF0) and data (IDEF1X) models as the "doctrine" against an operational scenario.
- Conceptually construct future instances from the objects identified in the domain model in the form of an operational scenario.
- Use the developed operational scenarios to assure the correctness of the domain model. For each operational scenario, analyze the domain model to verify that these specific future systems can be accommodated.
- Prototype the model to assure the execution of the domain model.
- Develop a simulation model to refine the object model, to explore adaptation mechanisms, and to conduct "what-if" analyses.
- Check the classification terms, guidelines, and rationale and tradeoffs for understandability, accuracy, consistency, and conformance with the domain model diagrams and specifications.
- Discuss and explain any problems encountered during inspection, prototyping, or simulation with domain analysts. Review suggestions for correcting these problems with analysts in order to expedite the refinement of the domain model.

1. **DEFINE DOMAIN ARCHITECTURE (TENA)** -- This activity builds a high-level TENA that will be the framework for the rest of the domain design effort. Often more than one existing solution can satisfy the requirements and constraints of the domain. In that case, multiple initial TENA-like architectures can be proposed, to provide: a) the opportunity for the domain engineering group to chose the optimal solution, or; b) the possibility of more than one solution for a domain. The effort necessary to design one TENA that adapts to all possible needs may prove impracticable, especially for large domains.

    1. **SELECT STANDARDS/PROTOCOLS** --
    2. **DEFINE APPLICATIONS FRAMEWORK** --
    3. **VALIDATE TENA** -- The goal of this process is to ensure that the TENA created meets the model requirements, domain constraints, and provides a workable solution. This validation provides confidence and motivation to potential users to reuse the design.

1. **DEVELOP INTEGRATED VALIDATION PLAN** -- Draft and final documentation will be developed in order to address the plans & objectives of the TENA validation. This document will address program goals and establish the success criteria for the conduct of a proof of principle demonstration(s). TENA represents an effort to develop an architecture. Consideration must be given to the methodology used to demonstrate the architectural concepts, particularly those which are not well represented in other implementations. This task will consist of developing the methodology for demonstration and validation. There are many possibilities, network simulation, lab test, field exercises, etc. However, for planning purposes the TENA project will identify a limited concept evaluation opportunity in FY1996 (aimed at validating architectural concepts) and a user concept validation in FY1996.

2. **INSPECT/REVIEW COMPONENT VALIDATIONS --** At this time the TENA Concept Validation team will inspect and review the component validation plan to ensure the TENA meets requirements.

3. **SIMULATE INTEGRATED TENA --**

1. *PROTOTYPE INTEGRATED TENA* --

2. **DEFINE TENA STEP II** -- As the requirements and applications concept processes are completed, the bulk of the architectural definition must begin in earnest. This is the effort which will produce several TENA candidates for concept validation, team and industry review. The results of this exercise include almost complete TENA definitions, standards and protocols which will have a sufficient depth to enable meaningful review, simulation, and selected concept validation exercises.

3. **DEFINE TENA STEP III** -- The results of industry reviews, simulations, and selected testing of concept architectures result in the refinement and selection of the proposed TENA.

1. **TRANSITION DEFINITION--** Transition is required to ensure that the TENA architecture becomes instanciated in the T&E and training communities, it is also a CTEIP requirement, and as such proper transition planning and definition need to be incorporated in the TENA Project early on. Draft and final TENA implementation guide Transition Planning are the responsibility of the Air Force and will be lead by IPT leader, Mr. Steve Salas of Edwards Air Force base. Mr. Salas will be supported by members of the Program Management Office and by other Service representatives.

1. **TRANSITION PROCESS PLANNING--** This process will define the

objectives, constraints and requirements of the transition process.

2. **DEVELOP USER'S GUIDE** -- TENA has broad applicability across many programs and organizations. As a result, it is necessary to describe in one document how TENA relates to various programs, users and customers. A user's guide will provide direction to those who choose to implement TENA and is primary source of the TENA intellectual property documentation. It will complement the architecture description and provide guidance to implementers. In addition, it will include the type of information and definitions required by project managers to determine TENA applicability and TENA compliance.

    1. **BUSINESS MODEL DRAFT** -- Develop a draft TENA business plan which will include a business model. The business model will outline the steps required to plan, organize, schedule and conduct a test or training scenario in the TENA environment. The business plan will identify issues which T&E facility users, e.g., Acquisition managers, training coordinators, test directors, etc., Will need to plan for and schedule in the multi-site/service TENA environment. The plan will also provide a comprehensive description of the capabilities and conceptual applications for the TENA environment. This information will assist users in determining TENA test capabilities are required/desired for their program.

    2. **BUSINESS MODEL FINAL** -- Develop a final TENA business plan after the review of the draft plan.

3. **CTEIP TRANSITION** -- The Transition Plan will recommend the necessary T&E funding, logistics and infrastructure needed to continue the benefits of the TENA architecture into the future. The Transition Plan will also document the assets remaining in place from the concept validation. The Transition Plan will define the support required so that organizations implementing the TENA architecture will have a source of; expertise, reusable assets and updates to the basic architecture.

    1. **DEMONSTRATION ASSETS DRAFT --** A draft demonstration procedure will be developed to demonstrate the TENA concept.

    2. **DEMONSTRATION ASSETS FINAL --** A final demonstration procedure will be developed to demonstrate the TENA concept

4. **TENA IMPLEMENTATION PLAN** -- The implementation guide will define the process to be followed, including equipment, software, and network connectivity, for a test or training site or activity to become TENA compliant. An implementation guide will provide direction to those who choose to implement TENA. It will complement the architecture description and provide guidance for implementation. In addition, it will include the type of information and definitions required by program managers to determine TENA applicability and TENA compliance.

    1. **ROADMAP DRAFT** -- The road map will recommend, at a high level,

TENA compliance implementation strategy for the T&E and training sites. The road map will address a broad category of issues relevant to and affecting TENA. These include changing directions in national security strategies, national military strategies and derived doctrine and the effects of emerging technology. The recommendations will be based on the requirements development, selected applications concepts and the lessons learned from the concept validation events.

2. **ROADMAP FINAL** -- Develop a final TENA roadmap after the review of the draft plan.

APPENDIX A

<u>ACRONYMS</u>

4GL	Fourth Generation Language
AHP	Analytic Hierarchy Process
ARC	Army Reuse Center
ASD(C3I)	Assistant Secretary Of Defense For C3I
ASW	Anti-Submarine Warfare
BPIP	Business Process Improvement Program
BPM	Business Processing Model
C3I	Command, Control, Communications, And Intelligence
CASE	Computer Aided Software Engineering
CIM	Center For Information Management
CMM	Capability Maturity Model
CODS	Class/Object Design Specification
CORS	Class/Object Requirement Specification
COTS	Commercial-Off-The-Shelf
DASD(IM) Information	Deputy Assistant Secretary Of Defense For Management
DEP	Domain Engineering Process
DBMS	Database Management System
DDI	Director Of Defense Information

DFD	Data Flow Diagram
DISA	Defense Information Systems Agency
DoD	Department of Defense
DSSA	Domain-Specific Software Architecture
ERAD	Entity-Relationship-Attribute Diagram
ERD	Entity-Relationship Diagram
FAPM	Functional Activity Program Manager
FD	Functional Description
FIM	Functional Information Manager
GDM/LDM	Global Data Model/Logical Data Model
GOTS	Government-Off-the-Shelf
GUI	Graphical User Interface
HDC	High-Demand Category
HW	Hardware
ICAM	Integrated Computer Aided Manufacturing
IDEF	ICAM Definition Method
IDEF0	This IDEF version provides Business Process Models
IDEF1X	This IDEF version provides Business Data Models
IRS	Interface Requirements Specifications
MIL-STD	Military Standard
MIS	Management of Information Systems
OMT	Object Modeling Technique
OO	Object-Oriented
OOA	Object-Oriented Analysis
OOD	Object-Oriented Design
PC	Personal Computer
PDL Language	Program Design Language or Program Definition

PEO	Program Executive Office
PEO STAMIS	Program Executive Office for Standard Army Management Information Systems
POC	Point-of-Contact
POL	Petroleum, Oils, and Lubricates
PSL	Program Statement Language or Problem Specification Language
RAPID	Reusable Ada Products for Information Systems Development
SDLC	Software Development Life-Cycle
SEI	Software Engineering Institute
SQL	Structure Query Language
SRP	Software Reuse Program
SRS	Software Requirements Specification
STD	State Transition Diagram
STARS	Software Technology for Adaptable, Reliable Systems
SW	Software
TENA	Test and Training ENabling Architecture
TIM	Technical Integration Manager
TRM	Technical Reference Model

APPENDIX B

DEFINITIONS

This appendix identifies and describes basic concepts and jargon which are helpful in understanding the domain analysis and design process.

Abstraction

(1) The principle of ignoring those aspects of a subject that are not relevant to the current purpose in order to concentrate more fully on those that are [Oxford, 1991:2].

(2) A problem analysis principle in which a problem is analyzed in general, then refined into subproblems, all of which exhibit the traits of the original problem [DAVI90:372].

(3) A principle to manage complexity.

Adaptation Analysis

The identification of differences among application system in the domain [GILR89].

Aggregation

See Whole-Part Structure

Application

(1) A working system which provides a set of general services for solving some type of user problem [PRIE91a:91].

(2) Software designed to fulfill specific needs of a user [DOD92:3].

Application Domain

The knowledge and concepts that pertain to a particular computer application. Examples include battle management, avionics, and nuclear physics [STARS].

Architecture

(1) The high-level design, concerned with the main components of a system and their roles and interrelationships [OXFORD 1991:456].

(2) The specification of a system at a somewhat general level, including description from the user interface, memory organization and addressing, I/O operation and control, etc. The implementation of an architecture in members of a given computer family may be quite different, yet all the members should be capable of running the same program. Common architecture provides compatibility from the user's viewpoint [OXFORD 1991:18].

(3) Describes the nature, configuration, and interconnection of the major pieces of an area of interest.

(4) A set of software modules (usually defined hierarchically) and their interfaces that define the structure of the software defined during design [DAVI90:372].

Artifact

An object, a structure, or a product.

Assembly Structure

See Whole-Part Structure

Attribute

(1) A defined property of an entity or object [OXFORD 1991:26].

(2) Any property, quality, or characteristic that can be ascribed to a person or thing [Webster's].

Examples: name, address, account number, unit cost

Behavior

(1) Action and response to stimuli [Webster's].

(2) How an object acts and reacts, in terms of its state changes and message passing [BOOC91:80].

Behavioral Model

A template that specifies the expected behavior (the objects reaction to stimuli) for an instance of an object class [EMBL92:63].

Behavior Modeling

The objective of behavioral modeling is to understand and document the way each object in a system (domain) interacts, functions, responds or performs ... describes what an object must do within its environment ... records the object's perceived states, the conditions and events that cause it to change from one state to another, the actions it perform, the actions performed on it ... and exceptions to normal behavior and real-time constraints imposed on object behavior [EMBL92:60].

Bottom-up Approach

A domain analysis approach where low-level requirements, source code and documentation from existing systems is analyzed to produce a preliminary vocabulary, a taxonomy, a classification structure and standard description. Source code is reverse-engineered to recover a design based on the existing system (s). This approach is used when old product artifacts exist and developers may or may not be available [PRIE91a:19].

Class

A family introduced in the programming language, SIMULA. The class provides a form of abstract data type (see data abstraction): it also provides the basis of the concept of object that underlies object-oriented languages [Oxford 1991:71].

Examples: vehicle, aircraft, document

Classification

(1) A mapping of a collection of objects to a set of descriptive domain terms.

(2) The process of determining such a mapping.

Cohesion

A measure of the degree to which parts of program module are closely functionally related. *High* cohesion means that each part is directed toward and essential for that module to perform its required function, and that the module performs only that function. *Low* cohesion might be due to convenience grouping of function that are unrelated by function, timing, logic, procedure, or by sequence [Oxford 1991:78].

Commonality Analysis

The identification of similarities among application systems in the domain [GILR89].

Component

A component is one object or operation in the domain [NEIG81].

Concurrency

The progressing of two or more Activities in parallel.

Context

The interrelated conditions or environment in which the domain exits or occurs.

Context Diagram

The diagram that represent the interfaces at the highest, most abstract level of a domain. It shows the primary purpose (process) of the domain, all domain terminators (i.e., objects external to the domain under specification that communicate or otherwise interface with the domain), and all the information flows between those terminators and the domain [DAVI90].

Coupling

A measure of the strength of the interconnections between modules of a program. A high coupling would indicate strong dependencies between one module and another. Loose coupling allows greater flexibility in the design and better traceability, isolation and correction of faults [Oxford 1991:102].

Data Abstraction

The principle of defining a data type in term of the operations that apply to objects of the type, with the operations that apply to object of the type, with the constraints that the values of such objects can be modified and observed only by the use of the operations [Oxford, 1991:109].

Data Model

A term used in a variety of situations with data storage at either a logical or physical level, but usually the former. It normally implies a formally defined structure within which the data may be represented [Oxford 1991:116].

Examples: ERDs, ERADs, semantics nets.

Data Modeling

The objective of this modeling technique is to understand and describe the "entities" and their interrelationships within the context of the area of interest. The emphasis is to identify and understand data *before* worrying about processing it [WHIT89]. Data modeling lacks inheritance, service, and message concepts of the object-oriented methodologies [COAD91E].

Examples: entity-relationship modeling, entity-relationship-attribute modeling, information modeling and semantic net modeling.

Data Type

An abstract set of possible values and the operations that may be performed on these values of an object [BOOC90:519].

Domain

A set of current and future systems that shares a set of common requirements, capabilities, and data. A logical grouping of related functions and objects. Often referred to as problem domain, problem space, or problem area.

Domain Analysis

The domain engineering process that identifies, collects, organizes, and represents domain information based on the study of existing systems, requirement specifications, domain expertise, engineering knowledge, and emerging technology. The process by which information used in developing software systems is identified, captured, and organized with the purpose of making it reusable [PRIE91a:17].

Domain Analyst

A person who conducts the domain analysis process. Their primary responsibilities include: knowledge acquisition, analysis and organization of information, and generation of domain products.

Domain Artifact

Any type or piece of domain-relevant information or knowledge that can be used to determine how to implement software solutions to domain problems. Domain artifacts include, but are not limited to, common business practices in the domain, and regulations, policies and directives governing the domain. Artifacts may be acquired from existing system documentation and source code, technical literature and trade journals, system surveys and questionnaires, technology newsletters and conferences, and applicable domain expertise in engineering disciplines such as systems analysis and design.

Domain Classification Scheme

A framework for domain-relevant terms that provides a structure (guideline) for identifying and describing pertinent characteristics of the respective domain components.

Domain Design

The domain engineering process that uses domain analysis products to construct generic architectures that facilitate construction of new system in the domain.

Domain Engineering

The encompassing system engineering discipline that includes identification (domain analysis), design (domain design), and the construction (domain implementation) of reusable components to address the problems of system and subsystem development.

Domain Expert

A person who has a good understanding of the capabilities and requirements for a family of application (s) or encapsulated problem area of interest. Domain experts, including system users, developers and maintainers, are consulted to determine application boundaries, refine and organize domain-specific concepts and vocabulary, provide rationales for specification and implementation concepts, and validate domain products.

Domain Implementation

The domain engineering Activities that use the products of domain analysis and domain design to construct assets for future reuse.

Domain Model

Identifies entity classed, inter-entity relationships, and operations on entities, which are common to most systems in a given problem domain [Oxford 1991:142]. The domain model provides the generic specifications for systems in the domain. Specifications for specific systems in the domain can be instantiated from the domain model to adapt unique, local requirements more rapidly and correctly.

Domain Modeling

The software engineering activity that develops or constructs a representation of a problem domain that exhibits the essential properties and behavior of the problem domain.

Domain-Specific Software Architecture (DSSA)

A specification for assemblage of software components that are specialized for a particular class of tasks (domain); generalized for effective use across that domain; composed in a standardized structure (topography); and effective for building successful applications [METT92]. A DSSA provides the common characteristics for the preliminary design of systems and software in a domain.

Domain Specifications

(1) A precise statement of the effects that the systems of the domain are required to achieve. Specifications should focus on what is to be done, rather than how it is to be done.

(2) The elicitation, capture and expression of requirements for a system (domain)

comprising hardware, software, and people [Oxford 1991:431].

(3) A description of the components or objects of the domain in terms of structure, context, attributes and services.

Entity

Any item that can be named or denoted [Oxford 1991:156].

Entity-Relationship Diagram

See ERA Diagram.

Entity-Relationship Model

See ERA Diagram.

Entity-Relationship-Attribute Diagram

See ERA Diagram.

Entity-Relationship-Attribute Model

See ERA Diagram.

Encapsulation

A principle, used when developing and overall program structure, that each component of a program should encapsulate or hide a single design decision ... the interface to each module is defined in such a way as to reveal as little as possible about its inner workings [Oxford, 1991]. See Information Hiding.

ERD

See ERA Diagram.

ERA Diagram

A diagrammatic notation for describing and documenting data items and the relationships between data items, using an ERA model [Oxford 1991:159].

ERA Model

A model of a set of data relationships in terms of the entities, relationships, an attributes. Entities have attributes and have relationships with other entities [Oxford 1991:159].

Facet

(1) Any definable aspect that makes up a subject or object [Webster's].

(2) In terms of a component's classification, an attribute of the component (e.g., component type, function, object, language, data representation, unit type, certification level, and environment).

Facet Term

An attribute used to describe a facet of a software component (e.g., implementation, add, number, add, integer, subprogram, level_1, Sun/UNIX).

Finite State Machine (FSM)

A virtual machine that can be in any one of a set of finite states and whose next states and outputs are functions of inputs and current states only. It is often useful to describe the behavior of a complex system as if it were and FSM [DAVI90:374].

Functional Clustering

(1) Cluster Analysis: A technique for grouping a set of similar elements and/or functions on the basis of observed qualitative and/or quantitative measurements, usually several variables [Oxford 1991:73].

(2) Bottom-up Development: Initially, each object is classified according to one specific category. Then, these categories are grouped based on the sharing of object attributes and/or services. The process of grouping and regrouping objects continues until the classes and hierarchies are refined [AUER88].

Gen-Spec Structure

See Generalization-Specialization Structure

Generalization-Specialization Structure

A superset object class is called *generalization*, and a subset object class is called *specialization.* Example: a FA-18 aircraft is a specialization (subset) of the generalization (superclass) Aircraft. The subclass FA-18 inherits attributes and services from its superclass Aircraft.

High-Domain Categories of Components

Those elements identified by the domain expert as having the greatest potential for reuse.

Horizontal Reuse

Reuse opportunities shared across multiple domains.

Information Hiding

A principle, used when developing and overall program structure, that each component of a program should encapsulate or hide single design decisions ... that are particularly difficult or likely to change. This approach leads to modules that are readily understood, can be developed independently and are easier to change [Oxford 1991:220].

Inheritance

(1) In a hierarchy of objects, an object generally has a parent object (superclass) at the next higher level in the hierarchy and one or more child objects (subclass) at the next lower level. ...Inheritance is a means by which characteristics of objects can be replicated and instantiated in other objects. Inheritance is both static by abstract data type and dynamic by instantiation and value [Oxford 1991:222].

(2) A mechanism for expressing similarity among Classes, simplifying definition of Classes similar to one (s) previously defined. It portrays generalization and specialization, making common Attributes and Services explicit within a Class hierarchy [COAD91E].

Instance Connection

An instance connection is a model of problem domain mapping (s) that one object needs with other objects, in order to fulfill its responsibilities ... Instance connections model association [COAD91E:127]. This association depicts cardinality relationships between objects.

Interface

A point of interconnection between two different physical or logical units.

Knowledge Base

A logical collection of information in a particular domain that has been formalized in the appropriate representation with which to perform reasoning ... A dynamic knowledge base is used to store information relevant to solving a particular problem and varies from one problem-solving session to the next [Oxford 1991:245].

Message

A message is a specification of one of an object's manipulations ... procedures of an object are activated by messages sent to the object by another object ... the basic control structure is message passing [Oxford 1991:312].

Message Connection

A message connection models the processing dependency of an object, indicating a need for services in order to fulfill its responsibilities [COAD91E:149].

Model

(1) *M* is a model of a system *S* if *M* can be used to answer a well-defined set of questions about *S* to a tolerance adequate for stated purpose [DAVI90:376].

(2) Models are sets of representations that depict a condition or set of operations in the real world ... Differs from a list of descriptions, in that it also describes the interrelationships of the components [FREE91:389].

Modularity

The property of a system that has been decomposed into a set of cohesive and loosely coupled modules. Modularity is one of the fundamental elements of the object model

[BOOC90:515].

Multiple Inheritance

An inheritance where the object receives its elements from more than one other object.

Object

An object is a package of information and a description of its manipulation ... and object comprises a data structure definition and its defined procedures in a single structure ... objects are instances of a class, each instance having its own private instance variables ... Each object can have various attributes associated with it. Attributes can be local to that object or inherited from the parent object [Oxford 1991:312].

Examples: Employee - in a Personnel System

Account - in a Financial System

Object-Behavioral Model

See Behavioral Model

Object-Interaction Model

Depicts the communication between objects through message connections.

Object Model

The graphical representations of the class & object structures, their behavior and their interrelationships within the area of interest. The object model encompasses the principles of abstraction, encapsulation, modularity, hierarchy, typing, concurrency and persistence [BOOC90:25].

Examples: Coad Object Diagrams, Object-Relationship Model, Object-Behavior Model.

Object Modeling

The objective of object modeling is to understand and describe and environment in terms of its objects while embracing the concepts of abstraction, encapsulation, modularity, hierarchy, typing, concurrency and persistence.

Object-Oriented (Development)

(1) An approach to developing software where every component represents and object in the real world, its attributes, and its possible actions; objects can be grouped together into classes to facilitate attribute and action assignments [DAVI90:376].

(2) a software engineering paradigm emphasizing the principles of abstraction, encapsulation, modularity, hierarchy, typing, concurrency, and persistence [BOOC90:516].

Object-Oriented Analysis (OOA)

Same as object-oriented development but applied exclusively to analysis.

Object-Oriented Design (OOD)

Same as object-oriented development but applied to design [DAVI90:376].

Object-Relationship Model

See ERA Diagram.

Persistence

The property of an object by which its existence transcends time (i.e., the object continues to exist after its creator ceases to exist) and/or space (i.e., the object's location moves from the address space in which it was created). Persistence is one of the fundamental elements of the object model [BOOC90:517].

PDL See Program Design Language.

Precedented Domains

Domains that have existing software systems or subsystems which serve as examples for subsequent development of the same or analogous software systems.

Problem Space

The delineation of *what* is to be done to resolve the problem (i.e., what types of data describe the environment, *what* is to be accomplished, *what* are the end-user requirements, *what* are the constraints and limitations), without defining *how* to do it (i.e., the analysis).

Procedural Abstraction

The principle that any operation that achieves a well-defined effect can be treated by its users as single entity, despite the fact the operation may actually be achieved by some sequence of lower-level operations [Oxford, 1991].

Process

A stream of activity.

Process Model

a description (graphical and textual) of the flow of data through the system and the processing performed on the data. The emphasis is on inputs, outputs, processing, and the relationship between processes [WHIT89]. The data flow diagram is the most popular depiction of the process model.

Examples: Structured Systems Analysis and Design, Structured Analysis and Design

Technique, structured analysis and System Specification (i.e., most modern structured analysis techniques).

Process Modeling

The method of analyzing and describing a system or domain in terms of the operations and actions that convert input to output.

Program Design Language

A language, used for expressing program designs, that is similar to conventional high-level programming language but emphasizes structure and intention rather than the ability to execute programs expressed in the language.

Reuse

To put or bring into action or service again; to employ for or apply to a given purpose again [COAD91E:12].

Reusable Component

A software element (including requirements, designs, code, test data, etc.) capable of being used by a software development effort other than the one for which it was originally developed.

Reusable Resource

A resource that is not rendered useless by being used [Oxford 1991:392].

Remembrance

The ability to retain a value or state information for future use.

Reference Model

A representation that allows people to agree on definitions, build common understanding and identify issues for resolution. The model also provides a mechanism for identifying the key issues associated with applications portability, scalability, and interoperability [DISA92a].

Requirements Forecasting

The estimation of future requirements.

Semantic Net

A means of representing relational knowledge as a labeled, directed graph. Each vertex of the graph represents a relation between concepts ... a semantic net is sometimes regarded as a graphical notation for logical formulae [Oxford:408].

Services

A service is a specific behavior that an object is responsible for exhibiting [COAD91E:143].

Examples: Display_Menu, Calculate_Net_Pay, Print_Paycheck.

Solution Space

The delineation of *how* to resolve a problem (i.e., the design).

Software Architecture

A structural model representing the high-level design packaging of functions, data, their relations, and control, to support the implementation of applications in a domain.

Software Reuse

The process of implementing new software systems using existing software components and information.

State

(1) One of the possible conditions in which an object may exist, characterized by definite quantities that are distinct from other quantities; at any given point in time [BOOC90:518].

(2) A state represents an object's status, phase, situation, or activity [EMBL92:60].

State Transition

The process of changing the state of an object is called a transition. events and conditions cause the transition [EMBL92:61].

State Transition Diagram

A notation used to show the state space of an instance of a given class, the events that cause a transition from one state to another, and the actions that result from a state change [BOOC90:518].

Structure

(1) A manner of organization [Webster's, 1977].

(2) Structure is an expression of problem-domain complexity, pertinent to the system's responsibilities [COAD91E:79].

(3) The concrete representation of the state with any other object, although all objects of the same class do share the same representation of their state [EMBL92:518].

Examples: <u>generalization-specialization or class structures:</u>

Invoice:	Sales_Invoice

Accoounts_Payable_Invoice

Translation: Sales_Invoice and

Accounts_Payable_Invoice
are specialized types of
Invoice

whole-part or assembly structure:

Invoice: Invoice_Header

Invoice_Body

Translation: Invoice_Header and
Invoice_Body are
components (parts) of the
Invoice type

Subclass

A class that inherits from one or more classes (which are called its immediate superclasses) [BOOC90:518].

Subdomain

A subset (a partition) of the domain for which there is a defined, cohesive purpose (operation).

Subject

An abstraction of classes and objects used to manage complexity and navigate effectively within large OOA models.

Examples: User_Interface, Report_Writer

Superclass

The class from which another class inherits (which is called its immediate subclass) [BOOC90:518].

System

(1) Anything we chose to regard (a) as an entity and (b) as comprising a set of related components [Oxford 1991:455].

(2) [to place together (Greek)] A set or arrangement of things so related or connected as to form a unit or organic whole [Webster's].

Systems Families

A collection or grouping of interrelated software systems in the domain that share a set of common characteristics.

Terminators

External sources and destinations of data [DAVI90].

Top-Down Approach

Domain analysis is performed from requirements and high-level designs of current and new systems. These are analyzed for commonality [PRIE91a:19].

Triggers

The events and conditions that activate state transitions are called triggers [EMBL92:61].

Unprecedented Domain

A domain that lacks any prior software system development history on which to base new development efforts. System development in unprecedented domains generally rely on emerging technology [COHEN:5].

Validation

The process of evaluating software to determine compliance with specified requirements [MIL-STD-2167A].

Verification

The process of evaluating the products of a given software development activity to determine correctness and consistency with respect to the products and standards provided as input to that activity [MIL-STD-2167A].

Vertical Reuse

Reusing assets from a given application into other applications (as-is or with some modification) within the scope of the same domain.

Whole-Part Structure

A common type of object relationship. In this type of relationship set, an object (aggregate or superpart) is composed of other objects (components or subparts) [EMBL92:44].

Examples: An engine, a gas tank, a steering wheel

(components) *are parts* of a car (aggregate).

-

APPENDIX C

REFERENCES AND SUGGESTED READINGS

(* *Denotes discussion of a domain analysis methodology*)

APPL92 Corporate *Information Management, Process Improvement Methodology* For DoD Functional Managers, D. Appleton Company, Inc., 1992.

ARC92 Army Reuse Center, *Domain Definition Report*, Document No. 1213-65-210/3, Contract No: DAEA26-87-D-2001, September 24, 1992.

ARNO88 Arnold,S.P. and Stepoway, S.L., *The Reuse System*: *Cataloging and Retrieval of Reusable Software*, In Tutorial: *Software Reuse - Emerging Technology*, Tracz, W. (Ed.), IEEE Computer Society Press, Washington, DC, pp 138-14l, 1988.

ARAN91 Arango, G;, (Shlumberger Laboratory for Comp. Science), Prieto-Diaz, R. (S/w Productivity Consortium), *Domain Analysis Concepts and Research Directions*, *In Domain Analysis and Software Systems Modeling*, IEEE Computer Society Press, 1991.

ARAN89*	Arango, G., "Domain Analysis - From Art Form to Engineering Discipline, Proc. 5th International *Workshop Software Specification and Design*, CA Press,	Los Alamitos, California, PP 152-159, 1989.

ARAN88	Arango, Domain Engineering for Software Reuse, Technical Report RCS-RTP8827, Dept. of Information and Computer Science, in of California, Irvine, 1988.

AUER88	Auer, Ken and Adams, Sam. "Domain Analysis: Object Oriented Methodologies and Techniques", Knowledge System Corp., position paper for Domain Analysis Working Group Session at OOPSLA `88.

BAIL92a	Bailin, S.C., CTA Incorporated, *Katpur: Knowledge Acquisition for Preservation of Tradeoffs and Underlying Rationales*, 1992.

BA1L92b	Bailin, S.C., "Towards a Case Based Software Engineering Environment", Proceedings of the WISR `92 5th Annual Workshop on Software Reuse, Palo Alto, California, October 26-29, 1992.

BA1L91*	Bailin, S.C., CTA Incorporated, *"A Knowledge Oriented Approach to Domain Engineering", Symposium On Reuse and Re-engineering,* National Institute of Software Quality and Productivity, Nov. 1991.

BAIL89	Bailin, S.C., CTA Incorporated "An Object Oriented Requirements Specification Method", *Communications of the ACM*, Vol. 29, No. 7, July 1987.

BAUE93	Bauer, D., A Reusable Parts Center, *IBM Systems Journal,* 32(4), pp 620-624.

BOEH88	Boehm, Dr. Barry W, "*A Spiral Model of Software Development and Enhancement",* Tutorial: *Software Engineering Project Management,* IEEE, Washington, DC, 1988.

BOOC91	Booch, 6, *Object Oriented Design with Applications*, Benjamin/Cummings Publishing Company, Inc., ISBN 0- 80s3-00910, 1991.

BOOC84	Booch, G., *Software Engineering with Ada*, second edition, Benjamin/Cummings Publishing, 1987.

BOWE91	Bowen, G.M. "*An Organized, Devoted, Project-Wide Reuse Effort*", Tri-Ada Proceedings, Oct. 1991.

BRAU91	Braun, C., "Domain Specific Software Architectures - Command and Control", *Proceedings of the WlSR `91 4th Annual Workshop on Software Reuse*, 1991.

BUHR90	Braun, R., *Practical Visual Techniques in System Design with Applications to Ada,* Prentice Hall, 1990.

CAMP90	Daniel B. McNicholl, et. al., *Common Ada Missile Packages* -

Phase 3 (CAMP3): Developing and Using Ada Parts in Real-Time Embedded Systems, Technical Report FO8635-88-C-0002, McDonnell Douglas Missile Systems Company, St. Louis, MO, 27 April 1990.

COAD91a Coad, P., "Why Use Object-Oriented Development? A Management Perspective*", Journal of Object-Oriented Programming*, pp 60-61, Oct. 1991.

COAD9lb Coad, P, "Adding to OOA Results", *Journal of Object-Oriented Programming,* pp 64-69, May 1991.

COADBlc Coad, P., "00A & OOD: A Continuum of Representation", *Journal of Object-Oriented Programming*, pp 55-56, Feb. 1991.

COAD91d Coad, P., "New Advances in Object Oriented Analysis", *Journal of Object-Oriented Programming*, pp 44-49, Jan 1991.

COAD91e Coad, P. and Yourdon, E., *Object Oriented Analysis*, Second Edition, Prentice Hall, Englewood Cliffs, NJ, 1991.

C0HE89 Cohen, J. and Hignite, B., "GTE Software Reuse for Information Management Systems*", Position paper for the Reuse in Practice Workshop*, Pittsburgh, PA, July 11-13, 1989.

COHEN* Cohen, S., *Process and Products for Software Reuse and Domain Analysis*, Software Engineering Institute.

C0LB92 Colbert, Edward, *Analysis of Current Object-Oriented Methods,* vol., Absolute Software Co., Inc., January 17, 1992.

COLB89 Colbert, E., (Absolute Software Co., Inc.), "The Object Oriented Software Development Method: A Practical Approach to Object-Oriented Development", *TRI-Ada Proceedings*, October 1989.

CONK88* Conklin, J., and Begeman, M. L. "IBIS: A Hypertext Tool for Exploratory Policy Discussions", Technical Report STP-082-88, MCC Corp., Austin TX, 1988.

DAVI90 Davis, Al, Software Retirements Analysis & Specification, Prentice Hall, Englewood Cliffs, NJ, 1990.

DISA93 *DISA/ClM Reuse Metrics Plan*, Draft, April, 1993.

DISA92a *Technical Reference Model for Information Management*, Version 1.3, Defense Information Systems Agency Center for Information Management, United States Department of Defense., July 31, 1992

DISA92b "DoD Software Reuse Initiative", *Domain Analysis Workshop Proceedings*, DISA/CIM/XRE, Falls Church, VA, September 21-22, 1992.

DISA94 *Procurement Domain Analysis and Design Pilot Project Report*, Version 2.4, Defense Information Systems Agency Center for Information Management, United States Department of Defense., October 1994.

DOD92 *Military Standard Software Development and Documentation*, MIL-STDSDD, September 4, 1992.

EMBL92 Embley, David W., Kurtz, Barry D., and Woodfield, Scott N., *Object-Oriented System Analysis: A Model-Driven Approach*, Yourdon Press, Englewood Cliffs,, NJ, 1992.

FREE9l Alan Freedman, *Computer Glossary*, Prentice Hall, Eng1ewood Cliffs, NJ, 1991.

FUTA88 Futatsugi, K, Goguen, J Messeguer, J and Okada, R., *Parameterized Programming in 0BJ2, In Tutorial: Software Reuse Emerging Technology*, Tracz, W. (Ed.), IEEE Computer Society Press, Washington, DC, pp 337-346, 1988.

G1LR89 Gilroy, K., Comer, E., Grau, and Merlet, P., *Impact of Domain Analysis on Reuse Methods*, Software Productivity Solutions, Inc., Final Report C04-087LD-0001-00, U.S. Army, Communications Electronic Command, Ft. Monmouth, NJ, November 1989.

GOLD89 Golden, WasiI, and Harken, *The Analytic Hierarchy Process*, New York, New York, 1989.

GUER87 Guerrieri, E., Classification Schemes and Reusability Measurements for Reusable Software Components", *Proceedings of the Workshop on Software Reuse*, Rocky Mountain Institute of Software Engineering, Boulder, CO, Oct. 14-16, 1987.

HARA86* Harandi, M. T. and Lubars, M. D., "Knowledge Base Software Development: A, Paradigm and a Tool, *Proceedings of the 1986 National Computer Conference*, AFIPS Press, Reston, VA, pp43-s0, 1986.

HOL190* Holibaugh, R., *Domain Analysis Method*, Rev Number 2.1, Software Engineering Institute, Joint Integrated Avionics Working Group (JIAWG), November 9, 1990.

ISCO91a* Iscoe, Neil, Arango, Guillermo and Williams, Gerry, "Domain Modeling for Software Engineering (EDS Austin Research Laboratory, Technical Report: AUS-0591-1, May 13, 1991)", *Participants Proceedings, Domain Modeling Workshop, 13th International Conference on Software Engineering*, Austin, TX, pp 1-4, 1991.

1SCO91b* Iscoe, Neil, Zheng-Yang Lui and Kar Yan Tam, "A Framework for Understanding and Discussing Domain Modeling (EDS Austin Research Laboratory, Technical Report: AUS-0s91-3, May 13, 1991)", *Participants Proceedings, Domain Modeling Workshop, 13th International Conference on*

Software Engineering, Austin, TX, pp 5-13, 1991.

ISCO92* Iscoe, Neil, Zheng-Yang Lui and Guohui Feng, "Using Application Knowledge in Software Synthesis: The Role of Domain Models," EDS Austin Research Laboratory, Technical Report: AUS-0992-1, September 15, 1992.

JAWO90 Jaworski, A., Hills, F., Durek, T.A., Faulk, S., and Gaffney, J., *A Domain Analysis Process*, Interim Report 90001-N, Software Productivity Consortium, Herndon, VA, January 1990.

KANG90 Kang, K.S., Cohen, J., Hess, W., Novak, W., and Peterson, S., *Feature-Oriented Domain Analysis (FODA) Feasibility Study*, CMU/SEI-90-TR-21. Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA, November 1990.

KANG89* Kang, R.C., "Features Analysis: An Approach to Domain Analysis", *Position Papers for the Reuse in Practice Workshop*, Software Engineering Institute, Pittsburgh, PA, July 1989.

KUNZ79 Kunz, W., and Rittel, H, "Issues as Elements of Information Systems, Working Paper 131", Institut fur Grundlagen der Planung, I A., Universitat der Stuttgart, 1979.

LEE89 Lee, K.J., and Rissman, M., "Application of Domain-Specific Software Architectures to Aircraft Flight Simulators and Training Devices", *Position paper for the Reuse in Practice Workshop*, Pittsburgh, PA, July 11-13, 1989.

LEWI91 Lewis, J.A., Henry, S.M., Rafura, D.G., and Shulman, R.S. (Virginia Tech), "An Empirical Study of the Object Oriented Paradigm and Software Reuse", OOPSLA 91, pp 184-196.

LOND86 London, Philip E. and Martin S. Feather. Implementing Specification Freedoms," Reading in *Artificial Intelligence and Software Engineering*, Charles Rich and Waters (eds.), Morgan Raufman Publishers, Los Altos, CA, 1986.

LOCK90 Lockemann, P.C. (University of Germany), *Object-Oriented Information Management*, Decision Support Systems s, pp 79-102, 1989.

LUBA88' Lubars, Mitchell D., "Wide-Spectrum Support for Software Reusability, *In Tutorial: Software Reuse Emerging Technology*,Tracz, W. (Ed.), IEEE Computer Society Press, Washington, DC, pp 27s-281, 1988.

LUBA89* Lubars, Mitchell D.and Harandi, M.T., "Addressing Software Reuse Through Knowledge Base Design,-" in Biggerstaff, Ted J and Perlis, Alan J., editors, *Software Reusability Volume II:Applications and Experience*, Addison Wesley, 1989.

LUBA91* Lubars, Mitchell D., "Domain Analysis and Domain Engineering in IDeA, in Prieto-Diaz, R., and Arango, G., *Domain Analysis and Software System Modeling*, IEEE Computer Society Press, Los Alamitos, CA, pp 163-178, 1991.

LUBA93 Lubars, Mitchell D., "Frameworks Versus Libraries: A Dichotomy of Reuse Strategies, *Proc. 6th Annual Workshop on Software Reuse*, Owego, NY, November 1993.

MART92 Martin, J., and Odel, J, *Object Oriented Analysis and Design*, Prentice Hall, Englewood Cliffs, NJ, 1992.

MCCA8s* McCain, R., "Reusable Software Component Construction a Product Oriented Paradigm", *Proc. 5th AiAA/ACM/NASA/IEEE Computers in Aerospace Conference*, Long Beach California, pp 12s-13s, 198s.

MCKA93* McKay, Dr. Charles, *Presentation of Recommended Approach to Risk Management for Reusable Objects Software Environment (ROSE) Project*, University of Houston - Clear Lake, TX, January, 1993.

METT92 Mettala, LTC Erik G., "Domain Specific Software Architecture" presentation, *Domain Analysis Workshop*,DARPA/SISTO, 21 Sep 1992.

MOOR89* Moore, J.M. & Bailin S.C., Domain Analysis: *Framework for Reuse Technical Report*, Computer Technology Associates, Rockville, MD, October, 1989.

NEIG84* Neighbors, J., "The Draco Approach to Constructing Software from Reusable Components", *IEEE Trans. on Software Engineering*, Vol. SE-10, No. 20, pp s64-s73, September 1984.

NEIG81 Neighbors, James M., *Software Construction Using Components*, University of California at Irvine Technical Report 160, 1981.

NORM91 Norman, J.R., "Object Oriented System Analysis: A Methodology for the 1990s", *Journal of Systems Mgmt*., pp32-40, Jul. 1991.

Oxford, 1991 Dictionary of Computing, Oxford University Press, 1991.

PERR88* Perry, J.M. (GTE Government Systems Corporation), and Shaw, M. (Carnegie Mellon University), "The Role of Domain Independence in Promoting S/W Reuse: Architectural Analysis of Systems", *In Position Papers for the Reuse in Practice Workshop, Software Engineering Institute*, Pittsburgh, PA, July, 1989.

PETE91a Petersen, G., *Requirements Analysis and Design Tool Report*, Software Technology Support Center (STSC), Hill Air Force Base, UT 84056, 1991.

PETE91b Petersen, G., *Software Reengineering Tool Report*, Software Technology Support Center (STSC), Hill Air Force Base, UT 84056, 1991.

PETE89 Peterson, A.S., "Coming to Terms with Terminology for Software Reuse", *Position paper for the Reuse in Practice Workshop*, Pittsburgh, PA, July 11-13, 1989.

P1PE92 Joanne Piper, "DoD Software Reuse Vision & Strategy", *CrossTalk Journal of Defense Software Engineering*, No. 37, pp2-8 Software Technology Support Center, Hill Air Force Base ,UT, October 1992.

PR1C92 Price, G., Daich, G., Murdock, D., Hidden E., *Test Preparation, Execution, and Analysis Tools Report*, Software Technology Support Center (STSC), Hill Air Force Base, UT 840s6, April 1992.

PR1E9la Prieto-Diaz, R., *Reuse Library Process Model*, Final Report, STARS Reuse Library Program, Contract F19628-88-D-0032, Task IS40, Electronic Systems Division, Air Force Systems Command, USAF, Hanscom AFB, MA 01731, March 1991.

PR1E9lb Prieto-Diaz, R., "A Domain Analysis Methodology", *Proceedings of the Workshop on Domain Modeling*, pp 138-140, 13th International Conference on Software Engineering, Austin, TX, May 13, 1991.

PR1E9lc Prieto-Diaz, R., and Arango, G., *Domain Analysis and Software Systems Modeling*, IEEE Computer Society Press, Los Alamitos, CA 1991.

PR1E88 Prieto-Diaz, R, and Jones, G.A., *Breathing New Life Into Old Software, In Tutorial: Software Reuse Emerging Technology*, Tracz, W (Ed.), IEEE Computed Society Press, Washington, D.C, pp 152-160, 1988.

PR1E87* Prieto-Diaz, R., "Domain Analysis for Reusability, *Proceedings of COMPSAC 87: The Eleventh Annual International Computer Software & Applications Conference*, Tokyo, Japan, pp 23-29, IEEE Computer Society, Washington, DC, Oct. 1987.

QUAN88 Quanrud, Richard B., *Generic Architecture Study*, Report 34s1-4-14/2, SofTech, Inc., January 22, 1988.

RUMB91 Rumbaugh, J et al.., *Object Oriented Modeling and Design*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1991.

SAAT91 Saaty; Andvargas, The Logic of Priorities and Logic Planning, RWS Publication, Pittsburg, PA, 1991.

SAAT90 Saaty, Andvargas, *The Analytic Hierarchy Process*, Pittsburg, PA 1990.

SE1D92 Seidewitz, E., Stark, M., *Principles of Object Oriented Software Development with Ada*, Millennium System Inc., 1992.

SHLA89* Shlaer, S. & Mellor, SJ. (Proj. Technology, Inc.), "An Object Oriented Approach to Domain Analysis", *ACM SIGSOFT, Software Engineering*

Notes, Vol. 14, pp 66-77, 5 Jul. 1989.

SITT92 Sittenauer, C., Olsem, M., Murdock, D., *Reengineering Tool Report*, Software Technology Support Center (STSC), Hill Air Force Base, UT 840s6, July 1992.

THEO93 Theofanos, Mary F. *A Comparative Review of the Domain Analysis Guidelines for the Defense Information Systems Agency*, Data Systems Research and Development Program, Technical Operations, Martin Marrietta Energy System, Inc., February 1993.

VIT90* Vitaletti, W. and Guerrieri, E. (SofTech), "Domain Analysis within the ISEC RAPID Center", *Proceedings of the Eighth Annual National Conference on Ada Technology*, Pages 460-470, US Army Communications Electronics Command, Ft. Monmouth, N.J., March 1990.

WALL92 Wallnau, Kurt (Paramax), "An Introduction to CARDS", *CrossTalk Journal of Defense Software Engineering*, No. 36, pp30-31 Software Technology Support Center, Hill Air Force Base UT, September 1992.

Webster's Webster's New Collegiate Dictionary, G. & C. Merriam Company, Springfield, Mass., 1981.

WHIT89 Whitten, J., Bentley, L., and Barlow, V., *Systems Analysis and Design Methods*, Irvin Press, Homewood, IL, 1989.

YOUR89 Yourdon, Edward., Modern Structured Analysis, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1989.

APPENDIX D

<u>HOW TO READ IDEF DIAGRAMS</u>

1. IDEF IS FOR UNDERSTANDING SYSTEMS VIA MODELING

IDEF is a technique that enables people to understand complex systems, and enables them to communicate their understanding to others.

As used here, a "system" may be defined as any combination of machinery (hardware) data and people, working together to perform a useful function. IDEF may be applied in planning, analysis, design, project management, or whenever documented understanding of a complex subject is useful. The result of applying IDEF is a "model" that shows, in a series of diagrams, the understanding gained.

2. "TOP-DOWN" ORGANIZATION OF THE MODEL

The diagrams in a model are organized in a hierarchic and modular "top-down" fashion, showing the breakdown of the system into its component parts. Application of IDEF starts with the most general or abstract description of the system to be produced. If this description is contained in a single "module", represented by a box, that box is broken down into a number of more detailed boxes, each of which represents a component part. The component parts are then detailed, each on another diagram. Each part shown on a detail diagram is again broken down, and so forth, until the system is described to any desired level of detail. Lower level diagrams, then, are detailed breakdowns of higher level diagrams. At each stage of breaking down the system, the higher level diagram is said to be the "parent" or overview of the lower-level "detail" diagrams.

3. DIAGRAMS ARE INDEXED BY NODE NUMBERS

In an IDEF diagram, the component parts are shown as numbered boxes. A diagram should have no more than six boxes. Each box is detailed in one diagram at the next lower level until a sufficient level of detail is reached.

The place of each diagram in a model is indicated by a "node number", derived from the numbering of boxes. For example, A21 is the diagram which details box 1 on the A2 diagram. Similarly, A2 details box 2 on the A0 diagram, which is the top diagram of the model. This hierarchy may be shown in an index of diagram names and their node numbers called "node index". The node index serves as a table of contents for a model.

The example shown below says that providing develop system (A0) is broken down into three sub-functions, A1 through A3. Design system (A2) is further broken down into three, more detailed sub-functions, (A21 through A23).

Levels of Diagrams
Corresponding Node Index

A-0	Develop System
A0	Develop System
A2	Design System

4. DIAGRAMS CONSIST OF LABELED BOXES AND ARROWS

In IDEF, boxes represent components in the breakdown, and arrows represent relationships between these components. Descriptive labels are written inside each box and along each arrow to describe their meaning. The notation is kept simple to permit easy reading with little special training.

The following is a sample IDEF diagram. Notice that the boxes represent the breakdown of activities or functions performed by the system and are named by verbs. Arrows, which represent objects or information are labeled with nouns. This emphasizes system functions and is called actigram. It is the usual kind. When data or objects are emphasized the boxes are labeled with nouns and the arrows are verbs. This type of diagram is called a datagram.

5. BOX AND ARROW SYNTAX

The sample IDEF diagram shows that the descriptive names and labels convey the box and arrow contents to the reader.

In addition to its label, the side at which an arrow enters or leaves a box shows its role as an input, control, output, or mechanism for the box.

Arrows may branch or be joined. The branches may each represent the same thing, or different things of the same general type.

6. ARROWS SHOW THE CONNECTION BETWEEN PARENT AND DETAIL DIAGRAM

Some arrows show both their source and destination boxes on the same diagram, while other arrows have one end unconnected. The unconnected arrows represent inputs, controls, or outputs of the parent box. To find the source or destination of these unconnected arrows, the reader must locate the matching arrows on the parent diagram. All such unconnected arrows must continue on the parent for the diagrams to be complete.

Although arrow connections from parent boxes to detail diagrams may be obvious from the labels, a special notation allows readers to do the match quickly. The letter I, C, O, or M is written near the unconnected end of the arrow on the detail diagram, to identify that the arrow is shown as an Input, Control, Output, or Mechanism on the parent box. To pinpoint the arrow more precisely, this letter is followed by a number giving the relative position at which the arrow is shown entering or leaving the parent box, numbering left to right and top to bottom. For example, "C3" written on an arrow in the detail diagram indicates that this arrow is shown as the third control arrow entering the parent box. These identifications are written on the matching arrows of the detail diagram.

Using this letter/number matching scheme, an arrow may have a different (but compatible) descriptive arrow label on the parent diagram and the detail diagram, if appropriate.

Also, an arrow shown as control or as input on a parent diagram is not limited to the same role on a detail diagram (for example, C2 on the parent box appears as an input to box 1 on its detail diagram in the example below).

In very special cases, an unconnected arrow on a detail diagram has no matching arrow on its parent, or vice versa. In this case, the arrow head or tail is shown enclosed in parentheses.

Inputs (on the left) are transformed into outputs (on the right). Controls (on the top) govern the way the transformation is done. Mechanisms (on the bottom) indicate the means by which the function is performed. A "mechanism" (or support) might be a person or a committee or a machine or a process.

Arrows represent single things or general classes of objects or information. The arrow label describes what the arrow represents.

The arrow structure of an IDEF diagram represents a constraint relationship among the boxes. It does not represent flow of control or sequence. The arrows entering a box show all that is needed by the box to perform its function. Therefore, the box is constrained by its input and control arrows.

An output of one box may satisfy some or all of the input or control conditions required by one or more other boxes. It is not necessary that each and every box have input and control and output. Also, several boxes can be performing their functions simultaneously.

APPENDIX E

<u>TENA IDEF DIAGRAMS</u>

-

-

-

-

-

-

-

-

-

-

-

-

-
-
-
-
-
-
-
-
-
-
-
-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-
-
-
-
-
-
-
-
-
-
-
-

-

-

-

-

-

-

-

-

-

-

-

-

-
-
-
-
-
-
-
-
-
-
-
-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-
-
-
-
-
-
-
-
-
-
-
-

-

-

-

-

-

-

-

-

-

-

-

-

-
-
-
-
-
-
-
-
-
-
-
-

-
-
-
-
-
-
-
-
-
-
-
-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

APPENDIX F

<u>TOOL OVERVIEW</u>

APPENDIX G

<u>SYSTEM QUESTIONNAIRE</u>